2013-12

Optimal patrol to detect attacks at dispersed heterogeneous locations

McGrath, Richard G., Jr.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

DISSERTATION

OPTIMAL PATROL TO DETECT ATTACKS AT DISPERSED HETEROGENEOUS LOCATIONS

by

Richard G. McGrath, Jr.

December 2013

Dissertation Supervisor: Kyle Y. Lin

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
December 2013	Dissertation	

4. TITLE AND SUBTITLE

OPTIMAL PATROL TO DETECT ATTACKS AT DISPERSED HETEROGENEOUS LOCATIONS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Richard G. McGrath, Jr.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this dissertation are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A

14. ABSTRACT

We study a patrol problem where several patrollers move between heterogeneous locations dispersed throughout an area of interest in order to detect enemy attacks. To formulate an effective patrol policy, the patrollers must take into account travel time between locations, as well as location-specific parameters, which include patroller inspection times, enemy attack times, and cost incurred due to an undetected attack. We consider both random and strategic attackers. A random attacker chooses a location to attack according to a probability distribution, while a strategic attacker plays a two-person zero-sum game with the patrollers. In some cases, we can compute the optimal solution using linear programming. This method, however, becomes computationally intractable as the problem size grows. Therefore, our research focuses on developing efficient heuristics, based on aggregate index values, fictitious play, and shortest paths. Numerical experiments demonstrate that our heuristics produce excellent results with computation time orders of magnitude less than what is required to compute the optimal solution.

15. SUBJECT TERMS

Optimal patrol, Multi-agent patrol, Semi-Markov decision process, Sequential decision making under uncertainty

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	113	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

OPTIMAL PATROL TO DETECT ATTACKS AT DISPERSED HETEROGENEOUS LOCATIONS

Richard G. McGrath, Jr.
Commander, United States Navy
B.S., Massachusetts Institute of Technology, 1990
M.S., Stanford University, 1991
M.A., Naval War College, 2006

Submitted in partial fulfillment of the
requirements for the degree of
DOCTOR OF PHILOSOPHY IN OPERATIONS RESEARCH
from the
NAVAL POSTGRADUATE SCHOOL
December 2013

Author: Richard G. McGrath, Jr.

Approved by:

Kyle Y. Lin
Associate Professor of
Operations Research
Dissertation Supervisor

Michael P. Atkinson
Assistant Professor of
Operations Research

Timothy H. Chung
Assistant Professor of
Systems Engineering

Craig W. Rasmussen
Professor of
Applied Mathematics

Javier P. Salmeron
Associate Professor of
Operations Research

Approved by: Robert F. Dell
Chair, Department of Operations Research

Approved by: O. Douglas Moses
Vice Provost of Academic Affairs

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

We study a patrol problem where several patrollers move between heterogeneous locations dispersed throughout an area of interest in order to detect enemy attacks. To formulate an effective patrol policy, the patrollers must take into account travel time between locations, as well as location-specific parameters, which include patroller inspection times, enemy attack times, and cost incurred due to an undetected attack. We consider both random and strategic attackers. A random attacker chooses a location to attack according to a probability distribution, while a strategic attacker plays a two-person zero-sum game with the patrollers. In some cases, we can compute the optimal solution using linear programming. This method, however, becomes computationally intractable as the problem size grows. Therefore, our research focuses on developing efficient heuristics, based on aggregate index values, fictitious play, and shortest paths. Numerical experiments demonstrate that our heuristics produce excellent results with computation time orders of magnitude less than what is required to compute the optimal solution.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

List of Tables

List of Acronyms and Abbreviations

AOI Area of interest
CDF Cumulative distribution function
FP Fictitious play
IED Improvised explosive device
IHE Index heuristic epoch
IHT Index heuristic time
LB Lower bound
LBLP Lower bound linear program
NPS Naval Postgraduate School
OPT Optimal solution
RALP Random-attacker linear program
SALP Strategic-attacker linear program
SMDP Semi-Markov decision process
SP Shortest path
SPR1 Shortest path with one revisit
SPR2 Shortest path with two revisits
SPR3 Shortest path with three revisits
TSP Traveling salesman problem
UAV Unmanned aerial vehicle

THIS PAGE INTENTIONALLY LEFT BLANK

Executive Summary

Patrol problems are encountered in many real-world situations. Generally speaking, a patrol is the movement of a guard force through a designated area of interest for the purpose of observation or security. Patrols are often conducted by authorized and specially trained individuals or groups, and are common in military and law-enforcement settings. The use of patrols, instead of fixed, continuous surveillance, is often necessary because of real-world limitations on time and resources. Patrollers must operate with the intent of maximizing the likelihood of detection of adversaries, infiltration, or attacks. The objective in solving patrol problems is to determine the actions or policies that will maximize this likelihood.

This work is motivated by the need to provide for effective security, usually with limited resources, and often against very sophisticated and capable enemies. Not only does the solution to a patrol problem need to be mathematically sound, it also needs to be executable. Additionally, it is often important to ensure that the solution to a patrol problem incorporates sufficient randomization, and thus be unpredictable to potential adversaries.

In this dissertation, we consider a problem where multiple locations dispersed throughout an area of interest are subject to attack. An attack is considered to be any activity that the patroller wants to interdict or prevent, such as planting or detonating an explosive device, stealing a valuable asset, or breaching a perimeter. We consider two attacker behaviors: random and strategic. A *random attacker* chooses a location to attack according to a probability distribution, while a *strategic attacker* plays a two-person zero-sum game with the patroller.

The patrol models in the literature focus on the case where attacks may occur at any place within the entire patrol area. In some scenarios, however, attacks may occur only at specific locations within an area of interest. Motivated by these observations, we model travel times between locations and the inspection time at each location explicitly, which complements the works in the literature that typically divide a large patrol area into contiguous, equal-size subareas. To the best of our knowledge, this dissertation is the first to study patrols among dispersed heterogeneous attack locations.

We study three cases. First, we consider the case of a single patroller against random attackers. We determine the optimal solution by modeling the state space of the system as a

network and solve a minimum cost-to-time ratio cycle problem using linear programming. The linear program, however, quickly becomes computationally intractable for problems of moderate size, which in our experiments include problems with more than five patrol locations assigned to a single patroller. By using an argument involving a fair charge for a patrol visit, we develop an index for each patrol location as a function of the time since the last inspection. We develop and test two heuristic methods based on an aggregate index: the index heuristic time (IHT) method and the index heuristic epoch (IHE) method. With the IHT method, the patroller considers all paths and partial paths that can be completed during a predetermined look-ahead time window to find the path with the smallest aggregate index per unit time, and then moves to inspect the first location in that path. The IHE method works in a similar fashion. In this method, however, a patroller looks ahead over paths that consist of a specified number of patrol locations, regardless of the total time those paths will take, and visits the first location in the path with the smallest aggregate index per unit time. To the best of our knowledge, this dissertation is the first to utilize an aggregate index in a continuous-time problem. These two heuristics produce favorable results in our numerical experiments.

Second, we study the case of a single patroller against strategic attackers. By modifying the linear program in the previous case, we determine the optimal policy that minimizes the largest expected cost per attack among all locations. This solution is usually a randomized policy, where a patroller selects the next location to visit based on a probability distribution. Because the linear program quickly becomes computationally intractable for problems of moderate size, we develop a heuristic that treats each patrol pattern as a pure strategy and allows the patroller to develop a randomized strategy from several pure strategies. We study two methods to generate patrol patterns: the shortest-path (SP) and fictitious-play (FP) methods. The SP method uses a combinatorial selection of patrol patterns based on the shortest Hamiltonian cycle in order to minimize travel times between locations. The FP method is an iterative method that generates patrol patterns based on fictitious play; however, it uses considerably more computation time than the SP method. The SP method produces very favorable results in numerical experiments for several graph structures and sizes.

Finally, we study the case of multiple patrollers against strategic attackers, where several patrollers work together to patrol an area of interest. We present a heuristic method for the patrol team to develop a mixed strategy by choosing among several pure strategies.

We determine pure strategies using two methods: one based on set partitions and the other based on the shortest Hamiltonian cycle. In the set-partition method, the patrol team divides the patrol locations among the individual patrollers, with each patroller then independently patrolling an assigned subset of locations. We present a policy-improvement algorithm that generates effective set partitions based on the heterogeneous properties of each location. In the shortest Hamiltonian cycle method, each patroller uses the same patrol pattern at evenly spaced time intervals. We see favorable results in numerical experiments for several graph structures and patroller combinations, where a lower bound based on a linear program is used as a benchmark, since the optimal solution is not available in general.

In summary, this work provides efficient methods to determine effective and executable patrol policies that minimize costs incurred due to undetected attacks. These methods have been tested on several problem sizes and structures with very favorable results, and can be directly applied to many types of military and non-military patrol problems.

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

This dissertation would not have been possible without the support and efforts of many. I would first like to thank my advisor, Professor Kyle Lin, for his insight and guidance throughout this endeavor. I am grateful for having had the opportunity to work with him. I would also like to thank the members of my committee, Professors Mike Atkinson, Tim Chung, Javier Salmeron, and Craig Rasmussen. They, along with the entire faculty of the Naval Postgraduate School, have made my time here especially rewarding, challenging, and productive.

I would also like to thank the Sailors, Chief Petty Officers, and Officers of the United States Navy. I would not have had the opportunity to do this work were it not for their and their families' dedication and sacrifice. It is a privilege to serve with you.

To all of my past teachers, from the Milford, Massachusetts public schools to NPS and all of the stops that I have been fortunate enough to make in between: Thank you. May I serve my future students as well as you have served me.

On a more personal note, I must especially thank my wife, Jessica, for without her unwavering love and support throughout my entire career, none of this would have been possible. Any accomplishments that I have been fortunate enough to have achieved are not mine, but ours.

A big thank you also goes to my sons, Alexander and Zachary, who inspire me every day. I am proud to be your father, and I look forward to our next adventure together as a family.

I would be remiss not to mention my grandfather, Mr. Joseph DiAntonio. His life-long emphasis on the importance of family and service has set an example for me that I continually strive to live up to. The impression you have made on me cannot be overstated.

Finally, this dissertation is dedicated to the memory of my parents, Richard and Marcella McGrath—my first, best, and most influential teachers.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Background and Motivation

Patrol problems are encountered in many real-world situations. Generally speaking, a patrol is the movement of a guard force through a designated area of interest (AOI) for the purpose of observation or security. Patrols are often conducted by authorized and specially trained individuals or groups, and are common in military and law-enforcement settings. The use of patrols, instead of fixed, continuous surveillance, is often necessary because of real-world limitations on time and resources. Patrollers must operate with the intent of maximizing the likelihood of detection of adversaries, infiltration, or attacks. The objective in solving patrol problems is to determine the actions or policies that will maximize this likelihood. In most patrol problems, consideration must be made for the time required for a patroller to travel between specific locations within an AOI, and the time required to conduct an inspection in order to detect illicit activities at a particular location.

There are several military and non-military applications of patrol problems. Military applications include the routing of an unmanned aerial vehicle (UAV) on a surveillance mission or the conduct of ground patrols to interdict the placement of improvised explosive devices (IEDs). Non-military applications include the movement of security guards through museums or art galleries; police forces patrolling streets in a city; security officials protecting airport terminals; and conductors checking passenger tickets on trains in order to detect fare evaders.

This work is motivated by the need to provide for effective security, usually with limited resources, and often against very sophisticated and capable enemies. Not only does the solution to a patrol problem need to be mathematically sound, it also needs to be executable. Additionally, it is often important to ensure that the solution to a patrol problem incorporates sufficient randomization, and thus be unpredictable to potential adversaries. This work attempts to address these issues.

1.2 Scope of Dissertation

In this dissertation, we consider a problem where multiple locations within an AOI are subject to attack. A patroller (defender) is assigned to the area in order to detect attacks before they can be completed. An attack is considered to be any activity that the patroller wants to interdict or prevent, such as planting or detonating an explosive device, stealing a valuable asset, or breaching a perimeter. The patroller moves between locations and conducts inspections at those locations in order to detect any illicit activity. A specified travel time is required for movements between locations. It then takes the patroller an additional specified amount of time to inspect a new location after he arrives. At the end of the time required to complete an inspection, the patroller can move to any other location in the area.

We explicitly model the patrol problem on a graph, where potential attack locations are represented by vertices. We consider the inclusion of inspection times at each vertex and travel times for the patroller to move along edges between vertices in the graph. We consider this problem in continuous time and structure the patrol model on a complete graph, where the edge length represents the travel time between each pair of vertices.

The time at which an attacker arrives at a location to conduct an attack is random, and occurs according to a Poisson process. When an attacker arrives at a location he begins an attack immediately. The time required to complete an attack is random, with a probability distribution that is known to the attacker and the patroller. The patroller detects any ongoing attacks at a location at the end of his inspection. We consider an attacker to be detected if both the patroller and attacker occupy the same location at the end of the patroller's inspection. The amount of time it takes to complete an attack, as well as the amount of damage that an undetected attack will cause, is specific to each location.

The patroller's objective is to determine a path of locations to visit and inspect that will minimize the long-run cost incurred due to undetected attacks. For instances where the cost of an attack is the same at all locations, this objective is equivalent to maximizing the probability of detecting an attack.

We consider three patrol models that are closely related:

1. *A single patroller against random attackers*: In the random-attacker case, an at-

tacker will choose a location to attack according to a probability distribution that is known to the patroller. This situation may occur when there is intelligence available regarding potential enemy attack locations. It may be possible from this intelligence to assign a likelihood of attack to specific locations.

2. *A single patroller against strategic attackers*: In the strategic-attacker case, an attacker will actively choose a location to attack in order to inflict the maximum expected damage. Conversely, the patroller seeks to conduct his patrol so as to sustain the least expected damage. This situation may occur with a more capable or better-resourced enemy, who can analyze the expected damage among several attack locations.
3. *Multiple patrollers against strategic attackers*: In this case, we extend the idea of a single patroller against strategic attackers by considering how a team of more than one patroller can effectively work together to minimize the expected damage from a strategic attacker.

In all of these cases, we assume that the attacker cannot observe the real-time location of the patroller. In other words, once an attacker initiates an attack, he will carry on with the attack until either completing the attack or getting detected. An attacker cannot time his attack, nor can he abandon an attack, based on real-time information about the patroller's location.

1.3 Literature Review

A patrol problem can be considered more generally as a type of search problem. Many types of search and patrol problems have been studied in diverse literatures. Early work on search theory focused on two general categories: one-sided search and search games. One-sided search refers to the assumption that a target does not respond to, or is even necessarily aware of, the searcher's actions. In this type of problem, the objective is often to maximize the probability of detection before a deadline, or to minimize the expected time or cost of a search (Benkoski et al. 1991).

The two-sided search problem, more commonly referred to as a search game, involves a searcher and a target who knows that he is being pursued. These type of search problems are generally formulated as game-theoretic problems. The information that the target has concerning the searcher will vary anywhere from complete information on the searcher's strategy to a complete lack of information (Benkoski et al. 1991). In these scenarios, a

searcher and target can be working in competition, whereby the target wishes to evade detection. Alternatively, a searcher and a target can be working in cooperation, such as a search and rescue scenario, where the objective for both is to minimize the time (or cost) of the search. In this dissertation, we examine the competition category of problems.

Patrol problems are a specific type of search problem. In a patrol problem, a searcher utilizes a patrol strategy to cover an area where an attacker or target may or may not be present (Alpern and Gal 2002). There are several types of game-theoretic patrol problems that relate to our work. An accumulation game is a type of patrol problem where a patroller visits several locations to collect materials hidden by an attacker. If the patroller finds a certain amount of the materials, he wins; otherwise, he loses (Alpern and Fokkink 2008, Kikuta and Ruckle 2002). An infiltration game is a type of patrol problem where an intruder attempts to penetrate an area without being intercepted by a patroller (Alpern 1992, Auger 1991, Garnaev et al. 1997, Ruckle 1983, Washburn and Wood 1995). An inspection game is a type of patrol problem where the patroller attempts to interdict an attacker during an attack (Avenhaus 2004, Zoroa et al. 2009). The infiltration and inspection game categories are most similar to the models that we examine.

There are several examples in search-game literature where the search area is modeled as a graph or network. Kikuta and Ruckle (1994) study initial point searches on weighted trees. Kikuta (1995) studies search games with traveling costs on a tree. Alpern (2010) examines search games on trees with asymmetric travel times. Basilico et al. (2009) present a deterministic patrolling strategy on a graph.

The works most closely related to this dissertation are those by Alpern et al. (2011) and Lin et al. (2013). Alpern et al. (2011) examine optimized random patrols where a facility to be patrolled is modeled on a graph with interconnected vertices representing individual locations within the facility. This work focuses on the case of strategic attackers, where an attacker actively chooses a location to attack, and assumes that the time to complete an attack is deterministic and is the same for all locations. Lin et al. (2013) examine a patrol problem on a graph with both random and strategic attackers. They use an exact linear program to compute an optimal solution. Since this method quickly becomes computationally intractable as a problem size increases, they introduce index heuristics based on Gittins et al. (2011) to determine a patrol policy. They use an aggregate index, where index values are accumulated as a patroller looks ahead into the future, to produce

effective patrol policies in a game-theoretic setting. Both of these works use discrete-time models, require the same inspection time at all locations, and prescribe an adjacency structure for their graphs—which puts constraints on how a patroller can move between locations.

In addition to the case of a single patroller, there has been much recent work on multi-agent patrol problems (Paruchuri et al. 2006, Paruchuri et al. 2007, Portugal and Rocha 2011). In the multi-agent case, a team of more than one patroller is assigned to a single patrol problem. The individual patrollers on the team may work cooperatively or independently to achieve a common goal. One popular method that has been studied in the multi-agent case is the use of patrol paths based on the traveling salesman problem (TSP), where patrollers follow the shortest Hamiltonian cycle in a graph in order to visit every location while minimizing the time between patrol visits to any one location (Chevaleyre 2004, Machado et al. 2003, Sak et al. 2008). Another method for multi-agent patrol on a graph involves partitioning, where vertices are put into subsets with each agent then patrolling his assigned subset of vertices using a TSP or closely related patrol path (Almeida et al. 2004, Elor and Bruckstein 2009). In this dissertation, we consider both the shortest-path and set-partition patrol methods in order to determine the best patrol policy in a game-theoretic setting.

1.4 Applications

The methods presented in this dissertation can be applied to several types of patrol problems, including how to determine the best movement of military units and assets to defend multiple locations when there is uncertainly regarding potential enemy attacks. One specific application in this area concerns the routing of a UAV conducting surveillance on several locations. UAVs are often well suited to a surveillance mission due to their ability to remain in an area for a long period of time. Like other assets, UAVs are often limited in availability and therefore must usually be used to monitor multiple locations. In this situation, a policy must be determined for how long a UAV remains at a location conducting surveillance before it moves to another location. An objective in this type of surveillance problem is to have the UAV in place at a location at the same time an enemy or attacker is present. If an attacker cannot observe the UAV surveillance, but instead will arrive at a specific location with some known or estimated likelihood, then the objective in routing the UAV is to maximize the likelihood of co-location. This type

of application is representative of the random-attacker problem that we examine in this dissertation.

Another example of a military application is a ground patrol conducted to interdict the placement of IEDs at specific locations within an AOI. Not only is it important to be able to detect the presence of IEDs once they are in place, but it is often tactically important to interdict the active placement of these devices in order to identify who is placing the IED. Therefore, an objective of the patroller in this scenario is to be at the same location as an attacker while he is emplacing the IED. Conversely, an attacker will choose his attack location such that he can avoid detection and incur the maximum expected cost or damage. When determining the maximum expected cost, an attacker must not only consider the actual cost or damage that will be incurred due to a successful attack, but also how likely the attack is to be successful. This type of application is representative of the strategic-attacker problem.

One non-military application for this type of patrol problem is the assignment of conductors to trains or other transportation systems, such as commuter ferries, in order to detect fare evaders (Avenhaus 2004, Tambe 2012). In such cases, a passenger is required to purchase a ticket for transport, but not all tickets are collected or checked prior to boarding a conveyance due to limited manpower and resources. Instead, a select number of conductors board certain trains randomly to check tickets. If we consider a fare evader to be an attacker for the purposes of our problem, an attack will be successful if he is able to ride a train without a ticket and complete his travel without being checked by a conductor. If a conductor is acting as the patroller, the time for him to conduct his inspection for tickets on a particular train car is analogous to the inspection time required at a location in our problem. Similarly, the time it takes for a conductor to move between train cars in order to begin another inspection for tickets is analogous to the travel time between locations.

Some additional examples of patrol problems that would benefit from the methods we present include the movement of security guards through museums or art galleries; police forces patrolling streets in a city; and security officials patrolling terminals in an airport.

1.5 Contributions

In this dissertation, we extend the work of Alpern et al. (2011) and Lin et al. (2013) by modeling travel times required for patrollers to move between locations that are subject to attack and inspection times at these locations. We examine the problem in continuous time for both random and strategic attackers and develop heuristics for determining optimal or near-optimal solutions. We develop heuristics for these problems because the methods used to determine an optimal solution quickly become computationally intractable as the size of the problem grows.

The patrol models in the literature focus on the case where attacks may occur at any place within the entire patrol area. In some scenarios, however, attacks may occur only at specific locations within an AOI. Motivated by these observations, in this dissertation we model the inspection time at each location and the travel time between locations explicitly, which complements the works in the literature that typically divide a large patrol area into contiguous, equal-size subareas. Specifically, to the best of our knowledge, this dissertation is the first to utilize an *aggregate index per unit time* in a continuous-time problem and the first under the condition of dispersed heterogeneous attack locations to develop, implement, and extensively test several exact and heuristic solution methods that produce effective patrol policies in both a random and game-theoretic setting.

In the case of a single patroller against random attackers, we present a linear program to determine the optimal patrol policy. This linear program is constructed as a minimum cost-to-time ratio cycle problem on a directed graph. We also present two heuristic methods based on the graph structure that utilize aggregate index values to determine a heuristic patrol policy.

In the case of a single patroller against strategic attackers, we present a linear program to determine the optimal patrol policy. This linear program is a modification of the minimum cost-to-time ratio cycle linear program used for a random attacker that minimizes the largest expected cost among all locations and provides a direct mapping to a mixed strategy. We also present two heuristic methods for this case. The first is a combinatorial method based on the shortest Hamiltonian cycle in the graph. The second is an iterative method based on fictitious play. We also present a linear program that provides a lower bound to the optimal solution, which helps evaluate our heuristic policy when the optimal solution is not available.

In the case of multiple patrollers against strategic attackers, we focus on developing heuristics because an optimal solution can be determined in only a few special cases. We present two methods for the patrollers to develop effective strategies. The first is based on set partitions, where locations are divided into subsets with each individual patroller executing his best patrol strategy on an assigned subset of locations, independent of the other patrollers. We present a one-step policy-improvement algorithm for use with the set-partition method that is based on the expected cost per attack in order to reassign vertices among the several patrollers. The second method is based on the shortest Hamiltonian cycle in the graph, where each patroller follows the same patrol pattern at evenly timed intervals. We incorporate both of these methods for the patrol team to develop an effective mixed strategy in a game-theoretic setting.

1.6 Organization

The remainder of this dissertation is organized as follows. Chapter 2 presents the case of a single patroller against random attackers. Chapter 3 presents the case of a single patroller against strategic attackers. Chapter 4 presents the case of multiple patrollers against strategic attackers. In Chapter 5, we present our conclusions and suggest areas for future research. The Appendix contains mathematical background and details on numerical experiments.

CHAPTER 2:
Single Patroller Against Random Attackers

In this chapter, we consider the case of a single patroller against random attackers. In this patrol problem, the patroller's objective is to determine a patrol policy that minimizes the long-run average cost due to undetected attacks. Section 2.1 introduces a patrol model on a graph, where an attacker chooses to attack a specific location based on a probability distribution that is known to the patroller. In Section 2.2, we present a linear program that determines the optimal solution to the patrol problem. Since the linear program quickly becomes computationally intractable as the size of the problem grows, we also present two heuristic methods for determining a solution in Section 2.3. We conduct extensive numerical experiments for several scenarios and present the results in Section 2.4. We make recommendations on how to best utilize the heuristic methods based on the experimental results.

2.1 Patrol Model

We consider a problem where multiple heterogeneous locations dispersed throughout an area of interest (AOI) are subject to attack. A patroller (defender) is assigned to patrol the area and inspect locations in order to detect attacks before they can be completed. An attack is considered to be any type of activity that the patroller wants to interdict and prevent, such as planting an explosive device, stealing a valuable asset, or breaching a perimeter. The patroller moves between locations and conducts inspections at those locations in order to detect illicit activity. We consider an attacker to be detected and his attack defeated if both the patroller and attacker occupy the same location at the end of an inspection.

We model this problem as a graph with n vertices, where each vertex represents a location that is subject to attack. We define a set of vertices $N = \{1, \ldots, n\}$ to represent potential attack locations. Figure 2.1 shows an example of a graph with five vertices. A random attacker will choose to attack vertex i with probability $p_i \geq 0$, for $i \in N$, and $\sum_{i=1}^{n} p_i = 1$. The time required for an attacker to complete an attack at vertex i is a random variable, which follows a distribution function $F_i(\cdot)$, for $i \in N$, that is known to the attacker and the patroller.

The patroller detects any ongoing attacks at a vertex at the end of an inspection. We assume that there are no false negatives; that is, the attacker will successfully detect all ongoing attacks at a vertex at the end of his inspection. An attack is considered to be unsuccessful if it is detected by the patroller. An attack is successful if it is completed before it is detected.

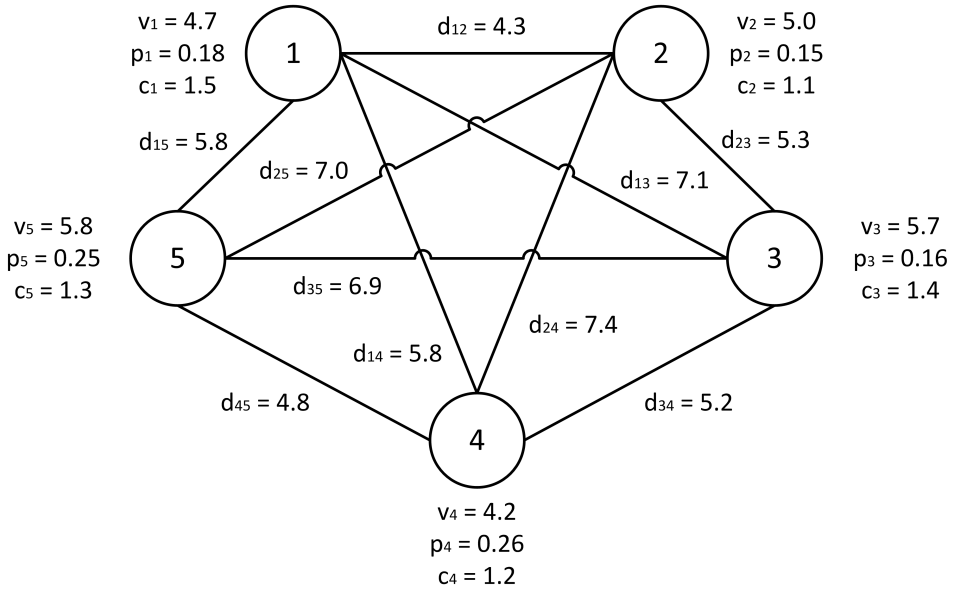

Figure 2.1: Example patrol graph with $n = 5$.

We assume that an attacker arrives at a location in the AOI to commence an attack according to a Poisson process with rate Λ. The Poisson process has stationary and independent increments, which implies that attacks are equally likely to occur at any time and that prior attacks do not help the patroller predict future attacks. Attackers arrive at a specific vertex i to begin an attack at a rate of $\lambda_i = p_i\Lambda$, for $i \in N$. These attacker arrivals at specific vertices constitute independent Poisson processes.

In most situations, the attacker arrival rate Λ is very small. In the formulation of our problem, the value of Λ is inconsequential because we ignore interruptions from attacks. In other words, several attackers can operate simultaneously on the graph, or even at the same vertex, with each acting independently. By minimizing the long-run cost rate, we also minimize the average cost from each attack with Λ acting as a scaling constant. Thus, the optimal solution does not depend on the value of Λ.

It takes a specified amount of time to travel between vertices and conduct inspections.

These times are fixed in our problem. The time required for a patroller to travel between vertices is denoted by an $n \times n$ distance matrix $D = [d_{ij}]$, for $i, j \in N$, where $d_{ij} \geq 0$ for all pairs of vertices $i \neq j$ and $d_{ii} = 0$. The time required for a patroller to complete an inspection at a vertex is denoted by $(v_1, \ldots, v_n)$. From these values, we construct an $n \times n$ transit time matrix denoted by $T = [t_{ij}]$, where $t_{ij} = d_{ij} + v_j$, to indicate the time required for a patroller to travel from vertex i to vertex j and complete an inspection at vertex j. The damage inflicted due to an undetected attack at a vertex is denoted by $(c_1, \ldots, c_n)$. An attack inflicts no damage if it is detected before it is completed.

The patroller travels between vertices in the graph and conducts inspections in order to detect attacks. A *patrol policy* consists of a sequence of vertices that the patroller will visit and inspect. We seek to determine the optimal patrol policy that minimizes the long-run cost rate incurred due to undetected attacks.

Fundamentally, the patroller is making a series of sequential decisions under uncertainty in order to determine a patrol policy. Decisions are made at decision epochs, which occur at a specific point in time (in this case at the end of an inspection). At each decision epoch, the patroller observes the state of the system as the amount of time elapsed since he last completed an inspection at each vertex. Based on this information, he chooses an action. The choice of action is which vertex to visit next. The action incurs a cost and causes the system to transition to a new state at a subsequent point in time. The cost incurred is the expected cost due to attacks that will be completed during the time it takes for the patroller to travel to and inspect the next vertex. At the end of the inspection time at the chosen vertex, the system will transition to a new state. At this point, the patroller reaches another decision epoch and the process repeats.

In our problem, we wish to determine the optimal choice of action for the patroller at each decision epoch. The essential elements of this sequential decision model are (Puterman 1994)

1. A set of system states.
2. A set of available actions.
3. A set of state-dependent and action-dependent costs.
4. A set of state-dependent and action-dependent transition times and transition probabilities.

We incorporate all of these elements into a sequential decision model in order to determine the optimal patrol policy.

2.2 Optimal Policy

In order to find the optimal solution to our patrol problem, we must determine a patrol policy that minimizes the long-run cost rate. To do so, we define a state space Ω that consists of all feasible states of the system. The state of the system at any given time can be delineated by

$$\mathbf{s} = (s_1, s_2, ..., s_n),$$

where s_i denotes the time elapsed since the patroller last completed an inspection at vertex i, for $i \in N$. Based on the assumption that a patroller detects all ongoing attacks at a vertex at the end of an inspection, the state of a vertex returns to 0 immediately upon completion of an inspection. Since we consider this problem in continuous time, the state of a vertex can assume any non-negative value. We write the state space of the system as

$$\Omega = \{(s_1, ..., s_n) : s_i \geq 0, \forall\, i \in N\}.$$

At the end of each inspection, the patroller reaches a decision epoch and will decide to stay at his current vertex to conduct an additional inspection or proceed to another vertex. The action space can be defined as

$$A = \{j : j \in N\}.$$

A deterministic, stationary patrol policy can be specified by a map π from the state space to the action space:

$$\pi : \Omega \to A.$$

This patrol policy is deterministic because, for any state of the system, a specific action is prescribed with certainty. It is stationary, or time-homogeneous, because the decision rules associated with a particular patrol policy do not change over time. For any given state of the system, the future of the process is independent of its past. The resulting state depends only on the action chosen by the patroller. If the patroller just inspected vertex k and next wants to inspect vertex j, that action will take time $d_{kj} + v_j$; and the

system that started in state $\mathbf{s}$ will transition to state

$$\tilde{\mathbf{s}} = (\tilde{s}_1, \tilde{s}_2, ..., \tilde{s}_n), \tilde{s}_j = 0; \tilde{s}_i = s_i + d_{kj} + v_j, \forall\, i \neq j\,.$$

In order to identify the vertex where a patroller has just finished an inspection and is currently located at a decision epoch, we define

$$\omega(\mathbf{s}) = \arg\min_i s_i,$$

since the state of the vertex where an inspection has just been completed will be 0 and the state of all other vertices will be greater than 0.

In our model, the times between decision epochs and state transitions are deterministic. They depend on previous system states and actions only through the current state of the system. We define

$$\tau(\mathbf{s}, j) = d_{\omega(\mathbf{s}),j} + v_j,$$

as the time between decision epochs and the time between state transitions, if the patroller decides to visit vertex j when the system is in state $\mathbf{s}$. At a decision epoch, the patroller will decide his action based only on the current state of the system. For these reasons, our model falls in the category of a semi-Markov decision process (SMDP).

The cost function for this SMDP can be calculated based on the distribution of time required to complete an attack $F_i(\cdot)$ and the cost c_i incurred due to a successful attack at vertex i. To illustrate how expected costs are incurred, suppose that the patroller has just finished an inspection at vertex k and the current state of the system is $\mathbf{s}$, where $\omega(\mathbf{s}) = k$. The patroller can then elect to travel to another vertex or remain at vertex k and conduct an additional inspection. There will be an expected cost incurred for each vertex in the graph based on the cost of a successful attack and the number of attacks expected to be completed at that vertex during the transition time between state $\mathbf{s}$ and state $\tilde{\mathbf{s}}$.

To determine the expected number of attacks that are completed at a particular vertex in a time interval, recall from Section 2.1 that the arrival of attackers at a vertex constitutes a Poisson process. Consider an attacker arriving to a vertex at time y after the last inspection was completed, and suppose that the patroller completes his next inspection

at that vertex at time s. The attacker will complete his attack if the attack time is no greater than $s - y$. Using Poisson sampling (see Proposition 5.3 in Ross (2010)), the number of successful attacks at vertex i will follow a Poisson distribution with expected value

$$\lambda_i \int_0^s P(X_i \leq s - y)\, dy = \lambda_i \int_0^s P(X_i \leq t)\, dt, \tag{2.1}$$

where X_i denotes the time required to complete an attack at vertex i, for $i \in N$.

If we know the expected number of attacks that will be completed at vertex i in a time interval, then we can determine the expected cost incurred at vertex i by multiplying (2.1) by c_i. Thus, the expected cost incurred at vertex i when the system is in state $\mathbf{s}$ and the patroller elects to transit to vertex j is

$$C_i(\mathbf{s}, j) = c_i \lambda_i \left(\int_0^{s_i + \tau(\mathbf{s},j)} P(X_i \leq t)\, dt - \int_0^{s_i} P(X_i \leq t)\, dt \right). \tag{2.2}$$

The cost at each vertex can be summed across all n vertices in the graph in order to determine the total expected cost when the system starts in state $\mathbf{s}$ and the patroller transits to vertex j. The overall cost function for this SMDP is

$$C(\mathbf{s}, j) = \sum_{i=1}^{n} C_i(\mathbf{s}, j). \tag{2.3}$$

As currently defined, the state space has an infinite number of states; however, in order to be able to compute the optimal policy, we need a finite state space. To do so, we assume that there is an upper limit on the attack time distribution at each vertex. Specifically, let B_i denote the maximum time required to complete an attack at vertex i. For the case where $s_i = S \geq B_i$, (2.2) becomes

$$C_i(\mathbf{s}, j) = c_i \lambda_i \left(\int_S^{S + \tau(\mathbf{s},j)} P(X_i \leq t)\, dt \right) = c_i \lambda_i (S + \tau(\mathbf{s}, j) - S) = c_i \lambda_i \tau(\mathbf{s}, j),$$

which remains a constant function over time for any state $s_i \geq B_i$. Therefore, once the state of a vertex has reached the bounded attack time, any additional expected cost will accrue at a constant rate. The bounded attack times allow us to restrict the state of a

vertex so that $s_i \leq B_i$, and the state space becomes

$$\Omega = \{(s_1, ..., s_n) : 0 \leq s_i \leq B_i, \forall\, i \in N\}.$$

We consider cases where the attack times at all vertices are bounded for the remainder of this dissertation. Thus, if the patroller has just inspected vertex k and next wants to inspect vertex j, the resulting state at the end of the inspection at vertex j is

$$\tilde{\mathbf{s}} = (\tilde{s}_1, \tilde{s}_2, ..., \tilde{s}_n), \tilde{s}_j = 0; \tilde{s}_i = \min\{s_i + d_{kj} + v_j, B_i\}, \forall\, i \neq j\,. \tag{2.4}$$

Using (2.4), we define a transition function to identify the resulting state if the patroller decides to visit vertex j when the system is in state $\mathbf{s}$:

$$\phi(\mathbf{s}, j) = \tilde{\mathbf{s}}.$$

The objective of the patrol problem it to determine a policy for the patroller that minimizes the long-run cost. Recall that the action space in this SMDP is finite because the number of vertices is finite. Therefore, by Theorem 11.3.2 in Puterman (1994), there exists a deterministic, stationary optimal policy. Thus, we only need to consider deterministic, stationary policies in our problem. We define

$$\psi_\pi(\mathbf{s}) = \phi(\mathbf{s}, \pi(\mathbf{s}))$$

as the resulting state if the patroller applies policy π when in state $\mathbf{s}$. We can define this function because the state transitions are deterministic. From an initial state $\mathbf{s_0}$, policy π will produce an indefinite sequence of states, $\{\psi_\pi^\kappa(\mathbf{s_0}), \kappa = 0, 1, 2, \dots\}$. This sequence must eventually visit some state for a second time since the state space if finite; and since the state transitions are deterministic under the same policy π, this sequence will then continue to repeat indefinitely. The sequence of vertices that correspond to this repeating cycle of states will constitute a *patrol pattern*.

We define V_i as the long-run expected cost rate at vertex i. If we apply the deterministic, stationary policy π to any initial state $\mathbf{s_0}$, then the long-run expected cost rate at vertex

i is

$$V_i(\pi, \mathbf{s_0}) = \lim_{\xi \to \infty} \frac{\sum_{\kappa=0}^{\xi} C_i(\psi_\pi^\kappa(\mathbf{s_0}), \pi(\psi_\pi^\kappa(\mathbf{s_0}))}{\sum_{\kappa=0}^{\xi} \tau(\psi_\pi^\kappa(\mathbf{s_0}), \pi(\psi_\pi^\kappa(\mathbf{s_0}))}. \tag{2.5}$$

We seek to determine the minimum long-run cost rate across all vertices, which will give the optimal solution

$$C^{\text{OPT}}(\mathbf{s_0}) = \min_{\pi \in \Pi} \sum_{i=1}^{n} V_i(\pi, \mathbf{s_0}), \tag{2.6}$$

where Π is the set of all feasible deterministic, stationary patrol policies. Dividing (2.6) by Λ will give the minimum average cost incurred for each attack.

We note that $V_i(\pi, \mathbf{s_0})$ depends on π and $\mathbf{s_0}$. However, in a connected graph, the optimal cost rate $C^{\text{OPT}}(\mathbf{s_0})$ does not depend on $\mathbf{s_0}$. Since determining the optimal patrol policy is equivalent to finding the optimal patrol pattern, we can develop a policy π in a connected graph that will produce any feasible patrol pattern from any starting state $\mathbf{s_0}$. Therefore, when we determine C^{OPT} in (2.6), it will be the same for all initial states since we minimize across all feasible patrol policies $\pi \in \Pi$. For the reminder of this dissertation we can drop the notational dependence of C^{OPT} on $\mathbf{s_0}$.

2.2.1 Linear Program Formulation

One method to solve this SMDP is to construct another graph that uses the state space of the system modeled as a network. To do so, we redefine the problem on a *directed* graph, $G(\mathcal{N}, \mathcal{A})$. Each *node* $k \in \mathcal{N}$ will represent one state of the system, and each *arc* $(k, l) \in \mathcal{A}$ will represent a feasible transition between states. This network will be of order $|\mathcal{N}| = |\Omega|$ and size $|\mathcal{A}| = |\Omega|n$. Each arc is assigned a transit time t_{kl} as determined by the vertex-pair specific distance and inspection times, where $t_{kl} = \tau(k, \omega(l))$; and cost c_{kl} as determined by the cost function (2.3), where $c_{kl} = C(k, \omega(l))$.

The objective is to find the directed cycle in the network with the smallest ratio of total cost to total transit time. This is a sufficient solution to the problem because any directed cycle in this network will constitute a valid patrol policy, regardless of the length of the cycle. This is an example of a minimum cost-to-time ratio cycle problem, also known as the *tramp steamer problem*, which is described in Section 5.7 of Ahuja et al. (1993).

To solve this problem, we formulate the following linear program, which we refer to as

the random-attacker linear program (RALP):

$$\min_{x} \sum_{(k,l)\in\mathcal{A}} c_{kl}x_{kl} \tag{2.7a}$$

$$\text{subject to} \quad \sum_{l|(k,l)\in\mathcal{A}} x_{kl} - \sum_{l|(l,k)\in\mathcal{A}} x_{lk} = 0, \qquad \forall\, k \in \mathcal{N} \tag{2.7b}$$

$$\sum_{(k,l)\in\mathcal{A}} t_{kl}x_{kl} = 1, \tag{2.7c}$$

$$x_{kl} \geq 0, \qquad \forall\, (k,l) \in \mathcal{A}\,. \tag{2.7d}$$

The variable x_{kl} represents the long-run rate at which the patroller uses arc (k,l). The objective function value in (2.7a) represents the long-run cost rate. The total rate at which the system enters state k must be equal to the total rate that the system exits state k, which is ensured by the network balance of flow constraint in (2.7b). For a single patroller, the rate that he uses arc (k,l) times the amount of time required to transit from node k to node l, indicates the fraction of time that he will spend on arc (k,l). The fractions of time must sum to 1, which is ensured by the total rate constraint in (2.7c). Finally, the long-run rate at which the patroller uses arc (k,l) cannot be negative, which is ensured by the non-negativity constraint in (2.7d).

The states on the optimal cycle directly correspond to vertices on the graph, which can be determined by the function $\omega(\mathbf{s})$. Thus, this linear program will produce a specific patrol pattern consisting of a repeating sequence of vertices for the patroller to visit and inspect. This patrol pattern represents the optimal solution to the patrol problem.

The number of decision variables in this linear program is $|\Omega|n$. The size of the constraint matrix is on the order of $|\Omega|$. The value of $|\Omega|$ grows as a function of the number of vertices in the graph, the attack time distributions, and the transit times.

2.2.2 Size of State Space

To understand the size of the state space, consider the case where the maximum attack time at all vertices is B, the travel time between all vertices is d, and the inspection time at all vertices is v. Define Z as

$$Z = \left\lceil \frac{B}{d+v} \right\rceil .$$

The number of states in the system for a graph with n vertices and $Z \geq n$ is given by

$$|\Omega| = \sum_{i=0}^{n-1} \binom{n}{1} \binom{n-1}{i} \binom{Z-1}{n-1-i} (n-1-i)!,$$

since for each state of the system there will be exactly one vertex in state 0, as indicated by the first term; i of the remaining $n-1$ vertices at the bounded attack time state B, as indicated by the second term; and each of the remaining $n-1-i$ vertices in a distinctive state between $d+v$ and $(d+v)(Z-1)$, as indicated by the third and fourth terms. Some examples of state space size are shown in Table 2.1. The number of states grows exponentially with the number of vertices, and grows even larger when combined with higher bounded attack times and shorter transit times.

Table 2.1: Examples of state space size.

n	B	d	v	Z	$\lvert\Omega\rvert$
5	9.8	1.0	0.2	9	16,965
7	11.5	1.2	0.3	8	$>$ 260,000
8	15.5	0.9	0.6	11	$>$ 20 million
12	18.3	0.8	0.8	12	$>$ 40 billion

Although we can compute the optimal patrol policy using linear programming, this method quickly becomes computationally intractable as the number of vertices increases and the ratio of the bound of the attack times to transit times increases. Hence, there is a need to develop efficient heuristics.

2.3 Heuristic Policies

In this section, we consider solutions based on index heuristic methods (Gittins et al. 2011). To begin, consider a special case of our problem when $v_i = 1$ and $d_{i,j} = 0$, for $i, j \in N$. This special case coincides with the model presented in Lin et al. (2013). By adding a Lagrange multiplier $w > 0$, Lin et al. (2013) show that the optimization problem can be broken into n separate problems, each concerning a single vertex. The Lagrange multiplier w can be interpreted as a service charge incurred for each patrol visit to a vertex. The objective is to decide how frequently to summon a patroller at each vertex in order to minimize the long-run cost rate due to undetected attacks and service charges. For a given state of the system, the solution to this problem can be used to determine an

index value for each vertex in the graph. We can develop a heuristic policy where, based only on the current state of the system, the patroller can choose to travel to and inspect the vertex that has the highest index value. We next explain how to extend this method to our patrol model.

2.3.1 Single Vertex Problem

We consider the problem at a single vertex where each visit from the patroller incurs a service charge $w > 0$. For a given value of w, our objective is to determine a policy that minimizes the total long-run cost rate due to undetected attacks and service charges. Generally speaking, a policy is a mapping from a state to an action. For the single vertex problem, the state of the system $s \geq 0$ is the amount of time since the patroller last completed an inspection at the vertex. The action space for the patroller simplifies to a binary decision: Inspect the vertex at time s or continue to wait.

Although the state space is infinite, the action space is finite for every $s \in \Omega$. Therefore, we only need to consider deterministic, stationary policies (Puterman 1994). In addition, since each inspection brings the state of the vertex back to 0, any deterministic, stationary policy reduces to the following format: Inspect the vertex once every s time units.

Recall from (2.1) that the number of successful attacks in the time interval $[0, s)$ between patroller inspections follows a Poisson distribution with expected value

$$\lambda \int_0^s P(X \leq t)\, dt\,.$$

Since each successful attack costs c, and a patrol visit costs w, the average long-run cost given a policy that inspects the vertex every s time units is

$$f(s) = \frac{c\lambda \int_0^s P(X \leq t)\, dt + w}{s}, \qquad s > 0.$$

For a given value of w, we find s in order to minimize $f(s)$. To minimize $f(s)$, we take the first derivative of $f(s)$, which gives

$$f'(s) = \frac{c\lambda}{s} P(X \leq s) - \frac{c\lambda}{s^2} \int_0^s P(X \leq t)\, dt - \frac{w}{s^2},$$

and set $f'(s) = 0$ to obtain

$$0 = c\lambda s P(X \leq s) - c\lambda \int_0^s P(X \leq t)\, dt - w.$$

We solve this equation for w as a new function of s:

$$W(s) = c\lambda \left(sP(X \leq s) - \int_0^s P(X \leq t)\, dt \right), \tag{2.8}$$

where $W(s)$ indicates the corresponding service charge such that it is optimal for the vertex to summon patrol visits once every s time units.

Since attack times at each vertex are bounded by a constant B, for cases where $s \geq B$ we note that

$$\begin{aligned} W(s) &= c\lambda \left(s - \int_0^s P(X \leq t)\, dt \right) = c\lambda \int_0^s P(X > t)\, dt \\ &= c\lambda \int_0^B P(X > t)\, dt = c\lambda \mathrm{E}[X], \end{aligned} \tag{2.9}$$

which remains the same for all $s \geq B$.

2.3.2 Index Heuristic Time Method

Since $W(s)$ represents the per-visit cost for the optimal policy that visits a vertex in state s, we can define an index value for vertex i based on (2.8) as

$$W_i(s) = c_i \lambda_i \left(sP(X_i \leq s) - \int_0^s P(X_i \leq t)\, dt \right),$$

if the last inspection at vertex i was completed s time units ago.

A straightforward heuristic method for the patroller at a decision epoch is to compute the index values based on the current state of each vertex and choose to visit the vertex that has the highest index value. This method will produce a feasible patrol pattern; however, it does not account for different travel times between vertices. To solve this problem, we develop methods for the patroller to look further ahead and compute *aggregate* index values before choosing which vertex to visit next. When computing an aggregate index in our continuous-time model, we consider the amount of time that different actions will

take. To do so, we select a fixed look-ahead time window δ and consider all feasible paths and partial paths beginning from the patroller's current vertex $\omega(\mathbf{s})$ that can be completed during time δ. We call this the index heuristic time (IHT) method. A value for δ is selected based on the structure of the graph and is discussed at the end of this section.

To illustrate the IHT method, we select a look-ahead window δ and examine an arbitrary patrol sequence over the next δ time units. For the time window $[0, \delta]$, let $S_i(t), t \in [0, \delta]$ denote the state of vertex i at time t. By definition, $S_i(0) = s_i$ and $S_i(t)$ increases over time at slope 1 until the patroller next completes an inspection at vertex i, when its value returns to 0. The aggregate index values accumulated at vertex i over the time window $[0, \delta]$ can be written as

$$\int_0^\delta W_i(S_i(t))\, dt, \qquad \forall\, i \in N.$$

For a given patrol sequence, the total index value for all n vertices over the time window $[0, \delta]$ is

$$\sum_{i=1}^{n} \int_0^\delta W_i(S_i(t))\, dt\,.$$

To determine a patrol pattern using the IHT method, we select a starting state of the system $\mathbf{s_0}$ and enumerate all possible paths over the next δ time units. We compute the total aggregate index value for each of these paths using (2.3.2), and choose the path with the highest *aggregate index value per unit time*. The first vertex along that path becomes the vertex that the patroller inspects next. We repeat this process using the new state of the system as the starting state, and continue to repeat the process to form a path of vertices. Recall that since the state space is finite, this sequence must eventually visit some state for a second time. The process terminates when a state repeats and a cycle has been found. The vertices corresponding to the states of the system on this cycle is the patrol pattern that results from using the IHT method.

In order to select a value for δ in the IHT method, we determine the average transit time r between all vertices in the graph as

$$r = \frac{\sum_{i=1}^{n} \sum_{j=1}^{n} (d_{ij} + v_j)}{n^2}. \tag{2.10}$$

We then choose a look-ahead time window in terms of multiples of r. For example, if we choose $\delta = 3r$ as a look-ahead window, then we are choosing an amount of time that on average will allow the patroller to visit any sequence of three vertices from his current vertex. We can choose a multiple of r more generally, such as $n/2$, which will on average allow the patroller to look ahead over about half the vertices in the graph from his current location. We make recommendations on how to select specific values for δ based on our numerical experiments. These recommendations are presented in Section 2.4.3.

Although we can choose any state from which to start the IHT method, for consistency in our numerical experiments we identify the vertex that has the maximum value of $W(s)$ when $s \geq B$, as defined in (2.9). We choose as $\mathbf{s_0}$ the state of the system where this vertex has just completed an inspection and the state of all other vertices is at the bounded attack time. In other words, we determine

$$k = \arg\max_{i \in N} \{c_i \lambda_i \mathrm{E}[X_i]\},$$

and select as $\mathbf{s_0}$ the state where $s_k = 0$ and $s_j = B_j$, for $j \in N, j \neq k$.

2.3.3 Index Heuristic Epoch Method

Instead of looking ahead for a fixed time period, as in the IHT method, we consider another heuristic which looks ahead for a fixed number of decision epochs. We call this the index heuristic epoch (IHE) method. To compute an aggregate index using the IHE method, we select a number of decision epochs η for the patroller to look ahead. The number η can be any positive integer value. For example, if we choose $\eta = 3$ as a look-ahead window, the patroller considers all paths of three vertices from his current vertex, since a decision epoch in our model occurs at the end of each inspection. As with the IHE method, we choose the path with the highest aggregate index value per unit time, and the first vertex along that path is the vertex that the patroller inspects next. We can also choose the look-ahead window more generally, such as $\eta = \left\lceil \frac{n}{2} \right\rceil$, which allows the patroller to look ahead over at least half the vertices in the graph.

We choose a starting state $\mathbf{s_0}$ for the IHE method using the same criteria as we did for the IHT method. We enumerate all feasible paths from $\mathbf{s_0}$ that consist of exactly η decision epochs and then proceed in the same manner as the IHT method described in Section 2.3.2 to determine a path of vertices based on the highest aggregate index value per unit

time, until a patrol pattern has been obtained.

2.4 Numerical Experiments

To test the IHT and IHE methods, we conduct several numerical experiments. We compare the results obtained from these heuristic methods with the optimal solution. We also report the computation time required. Based on these results, we make conclusions on the efficacy of the heuristic methods, as well as make recommendations for the selection of look-ahead parameters to be used in both the IHT and IHE methods.

As inputs for the problem, we use a probability vector $(p_1, \ldots, p_n)$ indicating the likelihood of an attacker to choose to attack a specific vertex; an attack time distribution parameter matrix; a vector $(c_1, \ldots, c_n)$ of the cost incurred due to a successful attack at each vertex; a distance matrix D of the time it takes for a patroller to travel between each pair of vertices; a vector $(v_1, \ldots, v_n)$ of the time required for a patroller to conduct an inspection at each vertex; and an overall attacker arrival rate Λ. Recall from Section 2.1 that the optimal solution does not depend on the value of Λ; therefore, without loss of generality, we set the overall attacker arrival rate to be $\Lambda = 1$ in our numerical experiments. We also set the cost incurred from a successful attack to $c_i = 1$, for $i \in N$, which allows the results to be interpreted as the long-run proportion of attackers that will evade detection.

We consider three general cases of patrol problems. In the first case, which we use as a baseline, the patroller spends about half of the time traveling and half of the time inspecting vertices. For this case, we choose average travel times that are comparable to average inspection times. In the second case, we choose average inspection times that are twice as long as average travel times. In other words, each vertex takes more time to inspect, but the vertices are closer together. In the third case, we choose average travel times that are twice as long as average inspection times. In other words, each vertex takes less time to inspect, but the vertices are farther apart.

All computations are done on a 64-bit Windows 7 desktop computer (Intel Core i7 860@2.8 GHz; 8.0 GB RAM). All linear programs that determine an optimal solution or a lower bound are implemented using GAMS 23.8.2 and are solved with CPLEX. MATLAB R2009b is used to implement and solve all heuristics.

2.4.1 Generation of Problem Instances

We conduct our numerical experiments on a graph with $n = 5$ vertices, which is a problem size that allows for the computation of the optimal solution. We choose parameters in order to generate and test cases where the optimal detection probability is in the neighborhood of 0.5. This is the case where the development of a good patrol policy can be most helpful.

To generate a random graph of n patrol locations for our experiments, let (X_i, Y_i) denote the Cartesian coordinate of vertex i, for $i \in N$, and draw X_i and Y_i from independent uniform distributions over $[0, 1]$. Letting d_{ij} denote the travel distance between vertices i and j, we compute

$$d_{i,j} = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}, \quad \forall\, i, j \in N.$$

The expected value of $d_{i,j}$ is $\mathrm{E}[d_{ij}] = 0.5215$ and the variance of $d_{i,j}$ is $\mathrm{Var}(d_{ij}) = 0.0615$. Details of how these values are determined are contained in the Appendix.

Based on this average distance and variance, we generate an inspection time at each vertex by drawing from a uniform distribution over $[0.3857, 0.6573]$. This distribution gives an expected inspection time of $\mathrm{E}[v_i] = 0.5215$, which is comparable to the average travel time between vertices. The variance of the inspection times is 0.00615, which is approximately $1/10$ of the variance of the vertex distance values. We choose these parameters in order to prevent very small inspection times at vertices, which could lead to excessively large state spaces and prevent the computation of an optimal solution.

For the attack time at each vertex, we use a triangular distribution. A triangular distribution requires three parameters: lower limit (minimum) a, upper limit (maximum) b and mode c, where $a < b$ and $a \leq c \leq b$. Additional details about triangular distributions are contained in the Appendix. We generate values for (a, b, c) independently from a uniform distribution over $[1.043, 4.172]$. This distribution gives a minimum attack time that is comparable to the average travel time between any two vertices plus the inspection time at the second vertex, which in this case is $0.5215 \times 2 = 1.043$. The expected value of this distribution is comparable to the time required for a patroller to travel and complete inspections over approximately half of the vertices in the graph, which for the case of $n = 5$ is $1.043 \times 5/2 = 2.6075$. From this minimum and expected value, we determine a

maximum attack time for use in our experiments as $2 \times 2.6075 - 1.043 = 4.172$. More generally, we can generate attack time distribution parameters from a uniform distribution on $[1.043, 1.043(n-1)]$ for problems with any number of vertices $n > 2$.

For the likelihood of an attacker to choose a vertex to attack, we create a probability vector $(p_1, \ldots, p_n)$. We spread 0.5 of the total attack probability equally across all n vertices and then randomly assign the remaining 0.5 probability. This ensures that the minimum probability of attack at any vertex is $0.5/n$, which will encourage a patrol policy that visits many or all of the vertices rather than completely excluding one or several vertices simply due to a low probability of attack. To create this vector, we generate n uniform random variables u_i on U[0, 1] and then normalize them so that $p_i = (0.5/n) + (0.5u_i / \sum_{j=1}^{n} u_j)$, for $i \in N$. In our experiments with $n = 5$, this ensures that each vertex has at least a 0.1 probability of selection for attack and no more than a 0.6 probability.

2.4.2 Baseline Problems

For our baseline problem, we consider the case where a patroller spends about half of the time traveling and half of the time inspecting vertices. We randomly generate 1,000 problem scenarios and determine the optimal solution using the RALP from Section 2.2 and a solution using the heuristic methods from Section 2.3. The RALP on average uses 5,920 decision variables and 7,105 constraints for a problem size with 1,184 states. The optimal solution takes on average 20.68 seconds to compute. We compare the solution obtained from the heuristic method to the optimal solution. For the look-ahead depth parameter δ used in the IHT method, we chose an initial value of $\delta = (n/2)r$, with r defined in (2.10) as the average transit time between vertex pairs in each problem instance. For $n = 5$, this starting value is $\delta = 2.5r$. We also test additional parameter values by increasing and decreasing the look-ahead depth in $0.5r$ increments.

As the IHT method looks further ahead, the computation time increases due to the higher number of paths that must be considered. Performance does not always improve when using deeper looks, and in many cases it may be worse. Two different look-ahead parameter values, $2.5r$ and $3r$ in the IHT method for example, may return the same patrol pattern or two distinct patrol patterns with different long-run cost rates. If the same problem is solved using multiple look-ahead parameters, we select the best solution that is obtained.

We consider single look-ahead parameter values and also consider sets of multiple look-ahead values in our numerical experiments. For the sets of multiple look-ahead values, we run the selected heuristic method for each individual value and then choose the patrol policy that yields the minimum cost, regardless of which specific look-ahead parameter produced that policy. This method tends to improve overall performance, but with a proportional increase in computation time based on the number and size of the look-ahead parameter values.

Results for the IHT method are shown in Table 2.2. When using a single look-ahead depth parameter, the best performance, as determined by the smallest excess over optimum for the mean and 90th percentile of problem instances, is obtained with a look-ahead time value of $\delta = 2.5r$. For the hybrid method of using up to three look-ahead parameters and then choosing the best patrol pattern, the best performance using similar criteria is obtained with a look-ahead depth set of $\{2r, 2.5r, 3r\}$.

Table 2.2: Performance of the IHT method on a complete graph with $n = 5$ vertices for 1,000 randomly generated problem scenarios with average inspection times comparable to average travel times, using the best solution that was obtained in each problem scenario for the indicated look-ahead depth parameter sets. Mean, 50th, 75th and 90th percentile performance is indicated as the percentage excess over the optimal solution.

IHT look-ahead depth	Percent over optimum				Time
(δ)	Mean	50th	75th	90th	(sec)
$2r$	3.31	0.38	4.13	8.65	2.19
$2.5r$	1.22	0.00	1.60	3.60	2.47
$3r$	1.36	0.00	1.34	5.51	3.64
$3.5r$	1.88	0.00	2.03	6.52	6.75
$4r$	3.26	1.24	5.61	7.96	18.22
$\{2r, 3r\}$	0.55	0.00	0.23	1.56	5.83
$\{2.5r, 3r\}$	0.62	0.00	0.49	2.15	6.11
$\{2r, 2.5r, 3r\}$	0.49	0.00	0.20	1.38	8.30
$\{2.5r, 3r, 3.5r\}$	0.49	0.00	0.23	1.39	12.86
$\{3r, 4r\}$	1.11	0.00	1.07	4.26	21.86
$\{2r, 3r, 4r\}$	0.54	0.00	0.23	1.56	24.05

We repeat the same experiments using the IHE method. For the look-ahead depth parameter η used in the IHE method, we chose an initial value of $\eta = \lceil \frac{n}{2} \rceil$. For $n = 5$ this starting value is $\eta = 3$. This indicates that, at each decision epoch, the patroller will consider all possible paths consisting of three decision epochs. We test additional

IHE depth parameter values by increasing and decreasing the look-ahead depth in $\eta = 1$ increments.

The IHE method is like the IHT method in that, as it looks further ahead, computation time increases due to the higher number of paths that must be considered. Similarly, the performance does not always improve when using deeper looks. For this reason, we test the IHE method using single look-ahead parameters and also using the hybrid method of comparing the results from multiple look-ahead parameters and selecting the best solution. Results are shown in Table 2.3. When using a single look-ahead depth parameter, the best performance, as determined by the smallest excess over optimum for the mean and 90th percentile of problem instances, is obtained with a decision epoch look-ahead value of $\eta = 4$. For the hybrid method of running the IHE method with several look-ahead parameters and then choosing the best patrol pattern, the best performance, as determined by a comparison of the excess over optimum and computation time required, is obtained using look-ahead depth sets of $\{2, 3, 4\}$ and $\{3, 4, 5\}$.

Table 2.3: Performance of the IHE method on a complete graph with $n = 5$ vertices for 1,000 randomly generated problem scenarios with average inspection times comparable to average travel times, using the best solution that was obtained in each problem scenario for the indicated look-ahead depth parameter sets. Mean, 50th, 75th and 90th percentile performance is indicated as the percentage excess over the optimal solution.

IHE look-ahead depth	Percent over optimum				Time
(η)	Mean	50th	75th	90th	(sec)
2	12.72	11.25	18.48	23.33	3.22
3	3.09	0.67	5.33	7.60	2.76
4	1.62	0.24	2.41	5.61	3.78
5	2.81	1.14	3.90	7.98	11.25
$\{2, 3\}$	2.87	0.28	4.32	7.36	5.98
$\{3, 4\}$	1.04	0.00	0.95	4.32	6.54
$\{2, 3, 4\}$	0.97	0.00	0.92	3.85	9.76
$\{4, 5\}$	1.30	0.00	1.49	4.36	15.03
$\{3, 4, 5\}$	0.89	0.00	0.63	3.68	17.79
$\{2, 3, 4, 5\}$	0.89	0.00	0.63	3.68	21.01

Performance of the IHT and IHE methods in the baseline case with a single look-ahead parameter is presented graphically in Figure 2.2. This figure shows a comparison of performance versus computation time required for different heuristic methods and look-ahead parameters. Although both methods perform well in the experiments, we tend to

see better performance using the IHT method in the single look-ahead parameter cases.

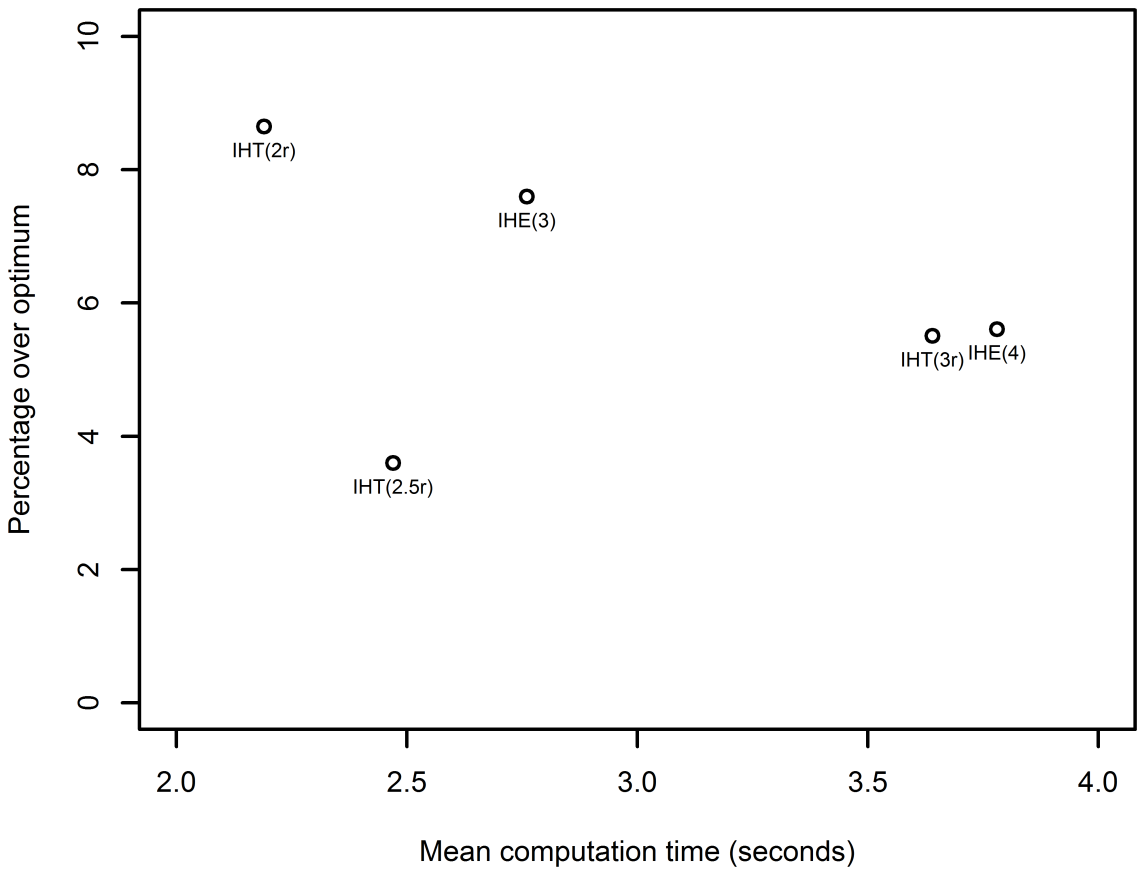

Figure 2.2: IHT and IHE 90th percentile performance with average travel times comparable to average inspection times.

In an effort to obtain the best possible results, we also use a hybrid set of look-ahead depth parameters that combine both the IHT and IHE methods. We selected various combinations of parameters to test based on the results from the individual IHT and IHE experiments. Results are shown in Table 2.4. Very good performance is obtained with a hybrid IHT look-ahead set of $\{2r, 2.5r, 3r\}$ and the performance improves when incrementally adding IHE look-ahead parameters.

Table 2.4: Performance of combined IHT and IHE methods on a complete graph with $n = 5$ vertices for 1,000 randomly generated problem scenarios with average inspection times comparable to average travel times, using the best solution that was obtained in each problem scenario for the indicated look-ahead depth parameter sets. Mean, 50th, 75th and 90th percentile performance is indicated as the percentage excess over the optimal solution.

IHT(δ) and IHE(η) look-ahead depth set	Percent over optimum				Time (sec)
	Mean	50th	75th	90th	
{IHT($2.5r$), IHE(3)}	0.88	0.00	0.95	3.45	5.18
{IHT($2.5r$), IHE(4)}	0.61	0.00	0.49	2.12	6.19
{IHT($2r, 2.5r, 3r$)}	0.49	0.00	0.20	1.38	8.30
{IHE(2, 3, 4)}	0.97	0.00	0.92	3.85	9.67
{IHT($2.5r, 3r$), IHE(3, 4)}	0.42	0.00	0.15	1.30	12.65
{IHT($2r, 2.5r, 3r$), IHE(2, 3, 4)}	0.30	0.00	0.00	0.92	17.89

Performance of the combined IHT and IHE methods in the baseline case for different look-ahead depth parameters is presented graphically in Figure 2.3. This figure shows a comparison of performance versus computation time required for different hybrid combinations of heuristic methods and look-ahead parameters. Both methods again perform well in the experiments, but we tend to see better performance using the IHT method in the hybrid set look-ahead cases, similar to the results from the single look-ahead parameter cases.

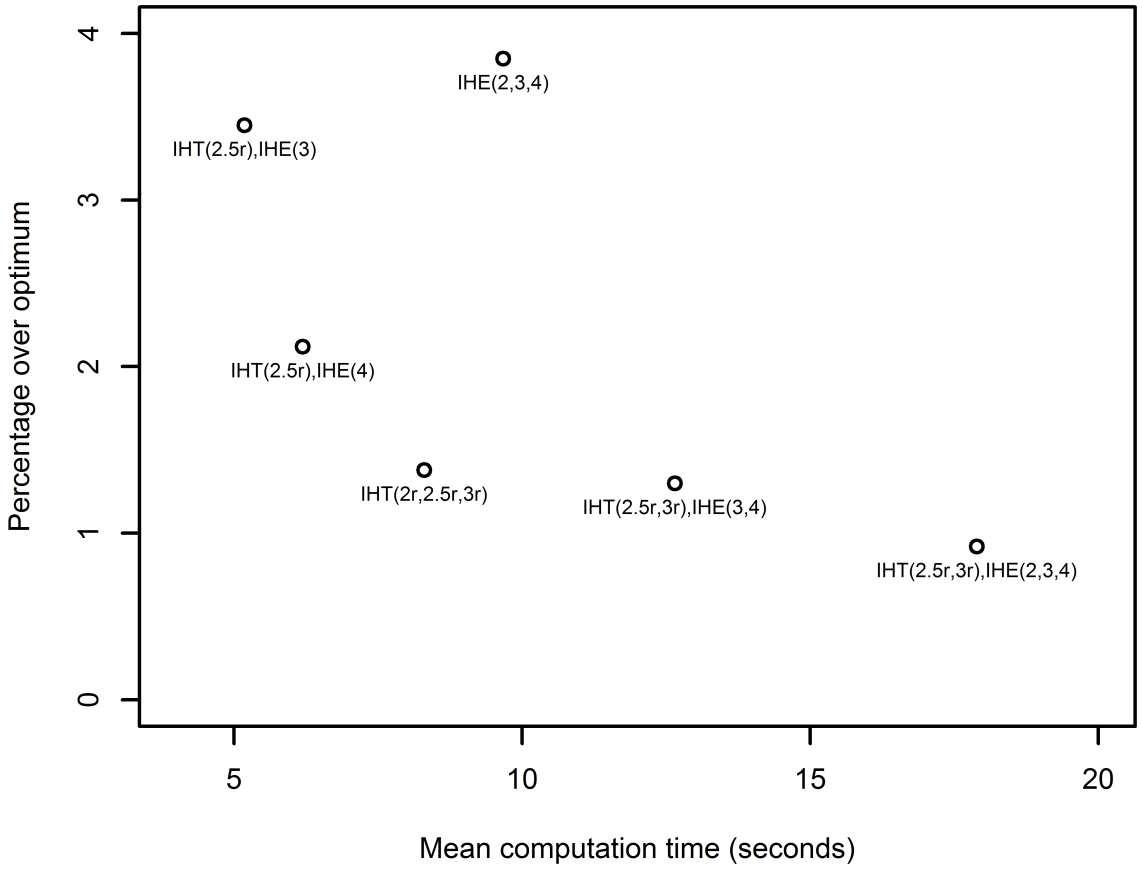

Figure 2.3: IHT and IHE hybrid 90th percentile performance with average travel times comparable to average inspection times.

2.4.3 Recommendations Based on Numerical Experiments

We see very favorable results using the IHT and IHE methods with many combinations of look-ahead parameters. In general, we have found that looking ahead over about half of the graph structure provides a good balance of performance versus computation time required. We recommend choosing look-ahead depth parameter values as a function of n, which represents the number of vertices that are assigned to a patroller.

Based on the experimental results, we recommend starting with the IHT method and using a look-ahead depth parameter value of $\delta = (n/2) \times r$, where r represents the average transit time in the graph. We then recommend adding additional looks using the hybrid method and selecting the best solution that is obtained. The total number of look-ahead depth parameters to use depends on the desired accuracy of a solution and computation time to be expended. Specifically, we recommend six prioritized look-ahead parameter values, each with a corresponding heuristic method, as presented in Table 2.5.

In a problem with $n = 5$, for example, after executing the heuristic method using IHT$(2.5r)$ we would next use IHT$(3r)$ and then continue in a similar manner until completing the desired number of looks. The IHE method is introduced at the fourth iteration of the heuristic method in order to complement the results obtained from using the IHT method.

Table 2.5: Prioritized heuristic methods and look-ahead depth parameters.

	Heuristic method and look-ahead depth parameter
1	$\text{IHT}\left(\frac{n}{2}r\right)$
2	$\text{IHT}\left(\frac{(n+1)}{2}r\right)$
3	$\text{IHT}\left(\frac{(n-1)}{2}r\right)$
4	$\text{IHE}\left(\lceil\frac{n}{2}\rceil\right)$
5	$\text{IHE}\left(\lceil\frac{n}{2}\rceil + 1\right)$
6	$\text{IHE}\left(\lceil\frac{n}{2}\rceil - 1\right)$

We test the prioritized look-ahead depth parameter set method using the baseline problem case. Results are presented in Table 2.6. The results indicate a steady improvement in performance, along with a corresponding increase in computation time required, as the number of looks increases. We observe that the heuristic method will return the optimal

solution in at least half of the problem instances when using a single look-ahead parameter IHT($2.5r$). The heuristic method will return a solution that is within 0.01 percent of optimal in at least 75 percent of the problem instances when using the fourth look-ahead set {IHT($2r, 2.5r, 3r$), IHE(3)}. Finally, we observe that the heuristic method will return a solution that is within 1 percent of optimal in at least 90 percent of the problem instances when using the fifth look-ahead set, {IHT($2r, 2.5r, 3r$), IHE(3,4)}.

Table 2.6: Performance of the IHT and IHE methods on a complete graph with $n = 5$ vertices for 1,000 randomly generated problem scenarios with average inspection times comparable to average travel times, using the best solution that was obtained in each problem scenario for the indicated look-ahead depth parameter sets. Mean, 50th, 75th and 90th percentile performance is indicated as the percentage excess over the optimal solution when using prioritized hybrid look-ahead depth sets as indicated. Mean time to compute the optimal solution is 20.68 seconds.

Heuristic	Percent over optimum				Time
set	Mean	50th	75th	90th	(sec)
1	1.22	0.00	1.60	3.60	2.47
2	0.62	0.00	0.49	2.15	6.11
3	0.49	0.00	0.20	1.38	8.30
4	0.37	0.00	0.01	1.29	10.96
5	0.30	0.00	0.00	0.92	14.67
6	0.30	0.00	0.00	0.92	17.89

These results are also presented graphically in Figure 2.4 to show the rate of improvement of the prioritized hybrid look-ahead depth sets as computation time increases. We observe the best rate of improvement in performance as a function of computation time required through the third look-ahead depth set {IHT($2r, 2.5r, 3r$)}. In Section 2.4.4, we test these recommendations further using several additional problem cases.

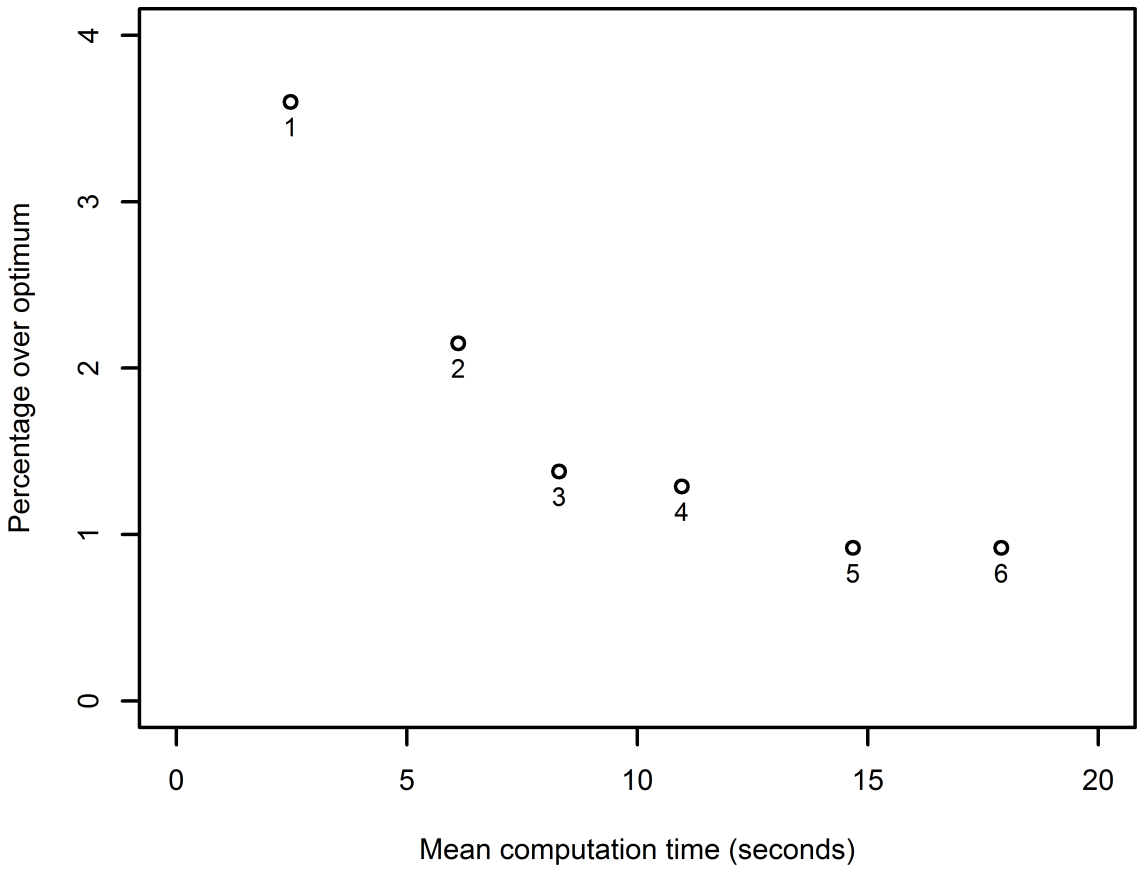

Figure 2.4: Combined IHT and IHE 90th percentile hybrid performance with average travel times comparable to average inspection times, using prioritized heuristic methods and look-ahead depth parameter sets.

2.4.4 Sensitivity Analysis

In addition to the baseline problems, we consider the case where a patroller needs to spend more time conducting inspections than he does traveling between vertices and the case where the patroller needs to spend more time traveling between vertices than he does conducting inspections. The problem cases considered in the numerical experiments are summarized in Table 2.7.

Table 2.7: Summary of numerical experiments for random attackers

Parameter	Case I	Case II	Case III	Case IV	Case V
Travel time	1×	1×	1×	2×	2×
Inspection time	1×	2×	2×	1×	1×
Attack time	1×	1.5×	1×	1.5×	1×
Mean travel time	0.5125	0.5125	0.5125	1.0430	1.0430
Mean inspection time	0.5125	1.0430	1.0430	0.5125	0.5125
Mean transit time	1.0430	1.5645	1.5645	1.5645	1.5645
Mean bounded attack time	3.2537	4.8805	3.2537	4.8805	3.2537
Mean number of states, $\|\Omega\|$	1,184	633	102	3,938	318
Mean number of decision variables	5,920	3,165	510	19,690	1,590
Mean number of constraints	7,105	3,799	613	23,674	1,909
Mean optimal long-run cost	0.3921	0.4200	0.5679	0.4617	0.5198
Mean optimal computation time (sec)	20.68	4.99	0.11	574.85	2.11

For the case where the average inspection times are longer than average travel times, we double the inspection times in the problem scenarios and run the experiment using both the linear programming and heuristic methods. We conduct these experiments with the original attack time distributions and also adjust the attack distributions as a separate case to maintain an overall probability of detection rate of approximately 0.5. We do this by increasing the attack time distribution parameters at each vertex by a factor of 1.5. The rest of the problem scenario parameters remain the same.

For the case where the average travel times are longer than average inspection times, we double the travel times in the problem scenarios and the run the experiment using both the linear programming and heuristic methods. We use the same original and adjusted attack distributions at each vertex that were used in the cases of increased inspection times as described above. The rest of the problem scenario parameters remain the same. Case I, the baseline case, had the lowest long-run cost on average. Case III generated

the smallest number of states and had the highest long-run cost on average. Case IV generated the largest number of states on average.

Results for problem cases II through V using the prioritized look-ahead parameter sets from Section 2.4.3 are presented in Table 2.8. In each of these problem cases, very favorable results were obtained using the recommended method of incrementally increasing the heuristic method and look-ahead parameter sets. We note that the heuristic performed slightly better in problem cases involving shorter travel times. The average computation time required in each case increases significantly as the average size of the state space grows. We particularly note this for problem Case IV, which had an average state space approximately three times larger that the baseline case, but required computation times that were approximately 25 times greater.

In general, the heuristic returns a solution within 0.01 percent of optimal in at least half of the problem instances using a single look-ahead parameter, IHT($2.5r$). The heuristic returns a solution within 0.01 percent of optimal in at least 75 percent of the problem instances using the third look-ahead set, $\{\text{IHT}(2r, 2.5r, 3r)\}$. Finally, we observe that the heuristic returns a solution within 1 percent of optimal in at least 90 percent of the problem instances using the sixth look-ahead set, $\{\text{IHT}(2r, 2.5r, 3r)$, IHE(2, 3, 4)$\}$. We also note in certain problem cases that this method may require more computation time than what is required to determine an optimal solution using the RALP.

Table 2.8: Performance of IHT and IHE methods for problem cases as indicated in Table 2.7, using prioritized look-ahead depth parameter sets. Performance is indicated as the percentage excess over the optimal solution.

Case	Time (sec) Optimal solution	Heuristic set	Percent over optimum				Time (sec) Heuristic solution
			Mean	50th	75th	90th	
I	20.68		See Table 2.6				
II	4.99	1	0.69	0.00	0.81	2.25	0.84
		2	0.35	0.00	0.13	1.47	1.95
		3	0.29	0.00	0.00	1.10	2.66
		4	0.26	0.00	0.00	0.84	3.45
		5	0.15	0.00	0.00	0.52	4.91
		6	0.14	0.00	0.00	0.36	5.68
III	0.11	1	0.99	0.00	0.94	2.99	0.09
		2	0.70	0.00	0.64	2.38	0.75
		3	0.41	0.00	0.01	1.31	0.81
		4	0.41	0.00	0.01	1.31	0.87
		5	0.35	0.00	0.00	1.12	1.08
		6	0.18	0.00	0.00	0.35	1.12
IV	574.85	1	2.03	0.01	2.48	6.90	51.12
		2	0.61	0.00	0.50	2.09	164.94
		3	0.41	0.00	0.01	1.21	203.11
		4	0.41	0.00	0.01	1.21	267.43
		5	0.39	0.00	0.00	0.82	320.75
		6	0.39	0.00	0.00	0.82	403.52
V	2.11	1	2.44	0.00	2.02	7.15	0.49
		2	1.06	0.00	0.67	3.97	1.96
		3	0.53	0.00	0.02	1.12	2.21
		4	0.44	0.00	0.00	0.96	2.43
		5	0.41	0.00	0.00	0.86	2.97
		6	0.41	0.00	0.00	0.86	3.18

CHAPTER 3:
Single Patroller Against Strategic Attackers

In this chapter, we consider the case of a single patroller against strategic attackers. Section 3.1 introduces a patrol model on a graph, where an attacker will actively choose a location to attack in order to incur the highest cost. In Section 3.2, we present a linear program that determines the optimal solution to the patrol problem. Since the linear program quickly becomes computationally intractable as the size of the problem grows, we also present heuristic methods for determining a solution in Section 3.3. In Section 3.4, we present a method to compute a lower bound for the optimal solution, which allows us to evaluate the heuristic methods when the optimal solution is unavailable. We conduct extensive numerical experiments for several scenarios and present the results in Section 3.5. We make recommendations on how to best utilize the heuristic methods based on the experimental results.

3.1 Patrol Model

We consider a patrol model similar to the random-attacker model presented in Section 2.1, except that in this case, an attacker will actively choose which vertex to attack in order to incur the highest expected cost. In other words, the attacker and the patroller play a simultaneous-move two-person zero-sum game where the attacker is trying to maximize the cost incurred due to a successful attack and the patroller is trying to minimize it. The patroller chooses how to patrol the graph while the attacker chooses which vertex to attack. Except for trivial cases, the optimal strategy for either player in a two-person zero-sum game is often a mixed strategy, which is a probability distribution on the set of a player's pure strategies (Owen 1995).

To formulate this problem, we modify the model that was used for the random-attacker case in Chapter 2. Recall from (2.5) that for a given patrol policy π, $V_i(\pi)$ is the long-run cost rate at vertex i. While the attacker is trying to maximize the expected cost incurred by choice of vertex to attack, the patroller is simultaneously trying to minimize it by choice of patrol policy. The patroller's objective function in this two-person zero-sum

game against a strategic attacker is

$$\min_{\pi \in \Pi^{\mathrm{R}}} \max_{i \in N} \frac{V_i(\pi)}{\lambda_i},$$

where Π^{R} is the set of randomized patrol policies.

3.2 Optimal Policy

It is possible to determine the optimal solution to this problem by formulating and solving a linear program. Recall the linear program from Section 2.2.1 that was used to find the optimal solution for the case of random attackers, where the objective function represented the overall long-run cost rate. In the case of strategic attackers, the objective is to minimize the largest expected cost per attack across each individual vertex, rather than the overall long-run cost rate for the entire graph.

To solve this problem, we again use the directed graph of the state space $G(\mathcal{N}, \mathcal{A})$, where each node $k \in \mathcal{N}$ represents one state of the system and each arc $(k, l) \in \mathcal{A}$ represents a feasible transition between states. Each arc is assigned a transit time t_{kl} as determined by the vertex-pair specific distance and inspection times, where $t_{kl} = \tau(k, \omega(l))$. Each arc is also assigned cost data that represents the expected cost incurred *at each vertex* when the system transitions from state k to state l. We write $c_{kl}^{(i)}$ as the expected cost incurred at vertex i for the state pair (k, l), as determined by (2.2), for $i \in N$.

If x_{kl} represents the long-run fraction of time that arc (k, l) is utilized during the patrol pattern, the long-run cost rate at vertex i is

$$\sum_{(k,l) \in \mathcal{A}} c_{kl}^{(i)} x_{kl}.$$

Dividing this total by the arrival rate of attackers at vertex i, we can define the zero-sum game between the patroller and strategic attacker as

$$\min_{x} \max_{i \in N} \sum_{(k,l) \in \mathcal{A}} \frac{c_{kl}^{(i)} x_{kl}}{\lambda_i}.$$

Note that $c_{kl}^{(i)} x_{kl}$ scales proportionately with λ_i, so the long-run average cost at vertex i does not depend on the value of λ_i. Hence for the rest of this section, we let $\lambda_i = 1$, for

all $i \in N$.

To determine the optimal solution for the strategic-attacker problem, we modify the linear program in Section 2.2.1 to minimize the largest long-run average cost per attack among all vertices, which we refer to as the strategic-attacker linear program (SALP):

$$\min_{x} z^{\mathrm{OPT}} \tag{3.1a}$$

$$\text{subject to} \quad \sum_{(k,l)\in\mathcal{A}} c_{kl}^{(i)} x_{kl} \le z^{\mathrm{OPT}}, \qquad \forall\, i \in N \tag{3.1b}$$

$$\sum_{l|(k,l)\in\mathcal{A}} x_{kl} - \sum_{l|(l,k)\in\mathcal{A}} x_{lk} = 0, \qquad \forall\, k \in \mathcal{N} \tag{3.1c}$$

$$\sum_{(k,l)\in\mathcal{A}} t_{kl} x_{kl} = 1, \tag{3.1d}$$

$$x_{kl} \ge 0, \qquad \forall\, (k,l) \in \mathcal{A}. \tag{3.1e}$$

In the optimal solution, the positive values of x_{kl} indicate the arcs that belong to the cycle with the lowest total cost per unit time. The states on these cycles directly correspond to vertices on the graph, which can be determined by the function $\omega(\mathbf{s})$. Therefore, an optimal mixed strategy patrol policy can be determined. For each state of the system, the patrol policy specifies the probability that the patroller will choose to move to each vertex. We map the solution from the linear program to a patrol policy using

$$p_{kl} = \frac{x_{kl}}{\sum_{l|(k,l)\in\mathcal{A}} x_{kl}}, \qquad \text{for} \sum_{l|(k,l)\in\mathcal{A}} x_{kl} > 0,$$

where p_{kl} is the probability that the patroller will choose to next go to vertex $\omega(l)$ when the system is in state k.

As the problem size grows, it quickly becomes computationally intractable to use this method. Therefore, there is a need for efficient heuristic policies.

3.3 Heuristic Policies

In this section, we consider heuristics to determine a strategy for the patroller. This method introduces a different kind of randomized strategy, by letting the patroller choose a patrol pattern from a predetermined set and repeat the patrol pattern indefinitely.

For the patrol problems we consider, there are an infinite number of feasible patrol patterns. As it would be impossible to consider an infinite number of patrol patterns, we propose a heuristic method to define a finite set of patrol patterns from which the patroller can select a mixed strategy. If it were possible to consider every feasible patrol pattern, then this method would find the optimal solution. Similarly, if we consider a finite subset of all the feasible patrol patterns, such that all patrol patterns that are part of the optimal solution are elements of that subset, then this method would also find the optimal solution.

We develop strategy reduction techniques that allow us to consider a comprehensive, but reasonable, number of patrol patterns for use in this heuristic method. To do so, we create a finite set S of feasible patrol patterns, ideally with elements that are identical or very similar to the patrol patterns that are part of the optimal solution. In the best case, S would contain all patrol patterns that are part of the optimal solution.

Once we determine a finite set of patrol patterns, $S = \{\xi_1, \xi_2, \ldots, \xi_m\}$, we formulate a different two-person zero-sum game between the attacker and the patroller in a standard matrix form. In this game matrix, row i corresponds to the attacker choosing to attack vertex i and column j corresponds to the patroller choosing patrol pattern ξ_j, for $i \in N$ and $j = 1, \ldots m$. A linear program can then be formulated to solve this two-person zero-sum matrix game (Washburn 2003). The solution to this game will provide a mixed strategy for both the attacker and the patroller, and the value of the game will be the expected cost due an undetected attack.

3.3.1 Patrol Cost Determination

For any feasible patrol pattern, we can determine the expected cost incurred at each vertex due to an undetected, and therefore successful, attack. We denote the expected cost at vertex j by ρ_j. These expected costs are used to populate the game matrix used in the heuristic method. There are three cases to consider when computing the expected cost at a vertex, which are based on the structure of the patrol pattern.

Case one occurs if the patrol pattern never visits vertex j. In this case, the expected cost for an attack on vertex j is c_j, due to the fact that if the attacker chooses to attack vertex j then the attack will always succeed. Thus,

$$\rho_j = c_j.$$

Case two occurs if the patroller visits vertex j exactly once during a patrol pattern of total time length τ. Recall from Section 2.2 that we can compute the expected number of successful attacks at vertex j when vertex j is inspected once every τ time units as

$$\lambda_j \int_0^\tau F_j(\tau - t)\, dt = \lambda_j \int_0^\tau F_j(s)\, ds\,.$$

Divide this by the expected number of attackers that will arrive at vertex j during time interval τ, which is $\lambda_j \tau$, to determine the probability of a successful attack:

$$\frac{\lambda_j \int_0^\tau F_j(s)\, ds}{\lambda_j \tau} = \frac{\int_0^\tau F_j(s)\, ds}{\tau}.$$

The expected cost at vertex j will therefore be the cost of a successful attack c_j times the probability of a successful attack:

$$\rho_j = \frac{c_j \int_0^\tau F_j(s)\, ds}{\tau}.$$

Case three occurs if the patroller visits vertex j two or more times during the patrol pattern. In this case, we break the patrol pattern into intervals based on each time the patroller returns to the vertex. If a patroller visits the vertex $m \geq 2$ times during a patrol pattern of total time length τ, we define t_1 as the time interval between the m-th (final) visit and the first visit to the vertex. The second interval t_2 is the time between the first and second visit. The last interval t_m is the time between visit $m-1$ and visit m. We compute the expected number of successful attacks at the vertex during each interval and divide that sum by the time to complete a full patrol cycle τ. Thus, the probability of a successful attack at vertex j, with $m \geq 2$ visits to vertex j, during a patrol pattern of total length $\tau = t_1 + t_2 + \cdots + t_m$ is

$$\frac{\lambda_j \int_0^{t_1} F_j(s)\, ds + \cdots + \lambda_j \int_0^{t_m} F_j(s)\, ds}{\lambda_j \tau} = \frac{\int_0^{t_1} F_j(s)\, ds + \cdots + \int_0^{t_m} F_j(s)\, ds}{\tau},$$

and the expected cost is

$$\rho_j = \frac{c_j \left(\int_0^{t_1} F_j(s)\, ds + \cdots + \int_0^{t_m} F_j(s)\, ds \right)}{\tau}.$$

3.3.2 Selection of Patrol Patterns

We consider two groups of patrol patterns to include in S. The first group is a combinatorial selection of patrol patterns based on the shortest Hamiltonian cycle in the graph. The second group is determined through an iterative method based on fictitious play.

Patrol Patterns Based on Shortest Path

Consider a case where the patroller chooses to use a single patrol pattern, or in other words, he uses a pure strategy. He would likely choose a pattern that visited each vertex at least once, since if he were to never visit a vertex, then an attack at that vertex would always be successful and would incur the full cost. Furthermore, he would likely try to minimize the time between inspections at each vertex.

To minimize the time between inspections at each vertex while visiting each vertex at least once during the patrol pattern, the patroller will follow a shortest Hamiltonian cycle in the graph. This patrol pattern is designated as the first element in the set S and we refer to it as the shortest-path patrol pattern. Finding the shortest-path patrol pattern is an example of solving a *traveling salesman problem*, as described in Section 16.5 of Ahuja et al. (1993), in which the vertices represent locations that are subject to attack and the weight on each edge is the time required to travel between those locations and complete an inspection at the arrival location.

From Section 3.3.1, the expected cost at vertex j using a shortest-path patrol pattern with total transit time τ is

$$\rho_j = \frac{c_j \int_0^\tau F_j(s)\, ds}{\tau}, \qquad \forall\, j \in N.$$

If a patroller were to use this patrol pattern as a pure strategy against strategic attackers, then the long-run cost of this policy is

$$V = \max_{j \in N} \rho_j,$$

since an attacker will employ his own pure strategy of always choosing to attack the vertex that incurs the highest cost.

Since we want to consider the option of a mixed strategy for the patroller, we must add additional patrol patterns to S. We start by considering subsets of the shortest-path

patrol pattern. Specifically, we consider n additional patrol patterns, which consist of the cycle where one vertex is skipped in the shortest-path patrol pattern and the patroller proceeds to the next vertex in the sequence. These are good patrol patterns to consider because they are consistent with the reasoning of using the shortest-path patrol pattern to minimize time spent on traveling, but they can also account for the heterogeneous qualities of potential attack locations. Due to differences among vertices in attack time distributions $F_i(\cdot)$ or cost incurred due to a successful attack c_i, a patroller may want to use a mixed strategy that periodically skips a visit to one or more vertices in order to occasionally direct more resources toward other vertices.

As an example, if the shortest-path patrol pattern in a graph with $n = 5$ vertices is $\{1-2-3-4-5-\}$, then the first subset of patrol patterns is

$$\begin{array}{llll} \{2- & 3- & 4- & 5-, \\ 1- & 3- & 4- & 5-, \\ 1- & 2- & 4- & 5-, \\ 1- & 2- & 3- & 5-, \\ 1- & 2- & 3- & 4-\}. \end{array}$$

For similar reasons, we also consider all paths of length $n-2$, where two vertices are removed from the shortest-path patrol pattern. In our example, there will be $\binom{5}{3} = 10$ of these patterns to consider:

$$\begin{array}{llllll} \{3- & 4- & 5-, & 2- & 4- & 5-, \\ 2- & 3- & 5-, & 2- & 3- & 4-, \\ 1- & 4- & 5-, & 1- & 3- & 5-, \\ 1- & 3- & 4-, & 1- & 2- & 5-, \\ 1- & 2- & 4-, & 1- & 2- & 3-\}. \end{array}$$

We continue this process by removing vertices until all subsets of the shortest-path patrol pattern that consist of only one vertex have been considered. For paths of length greater than three, the sequence of vertices can be reordered as required, so that the patroller will be utilizing the shortest Hamiltonian cycle within a particular subgraph of vertices. The total number of patrol patterns considered when using this method is $2^n - 1$. We refer to this set of patterns as the shortest-path (SP) patrol patterns.

In addition to the shortest-path patrol pattern and its subsets, we consider patrol patterns where the patroller chooses one vertex to visit twice during his patrol while visiting each remaining vertex only once. Ideally, we would choose the time for a revisit to a vertex in the patrol pattern such that the time between inspections is as close to even as possible. To determine these patterns, we continue to use the shortest-path patrol pattern as a baseline and insert a revisit to each vertex at all possible points in the pattern, such that the patroller does not complete a revisit to a vertex immediately after completing an inspection at that vertex. Using this method, we will consider an additional $n(n-2)$ patrol patterns. We refer to this set of patrol patterns as the shortest-path with one revisit (SPR1) patrol patterns.

To continue the example from above, for a graph with $n = 5$ vertices and shortest-path patrol pattern $\{1-2-3-4-5-\}$, the SPR1 set would consist of the following additional 15 patrol patterns

$$
\begin{array}{llllll}
\{1- & 2- & 1- & 3- & 4- & 5-, \\
1- & 2- & 3- & 1- & 4- & 5-, \\
1- & 2- & 3- & 4- & 1- & 5-, \\
1- & 2- & 3- & 2- & 4- & 5-, \\
1- & 2- & 3- & 4- & 2- & 5-, \\
1- & 2- & 3- & 4- & 5- & 2-, \\
1- & 3- & 2- & 3- & 4- & 5-, \\
1- & 2- & 3- & 4- & 3- & 5-, \\
1- & 2- & 3- & 4- & 5- & 3-, \\
1- & 4- & 2- & 3- & 4- & 5-, \\
1- & 2- & 4- & 3- & 4- & 5-, \\
1- & 2- & 3- & 4- & 5- & 4-, \\
1- & 5- & 2- & 3- & 4- & 5-, \\
1- & 2- & 5- & 3- & 4- & 5-, \\
1- & 2- & 3- & 5- & 4- & 5-\}.
\end{array}
$$

Similarly, we can continue this method of generating additional patrol patterns based on the shortest-path patrol pattern by allowing multiple revisits to a vertex. We consider the case of the shortest path with two revisits (SPR2) by starting with the SPR1 patrol patterns and, for each of these patrol patterns, conducting an additional visit to each vertex. We consider paths that revisit all combinations of two vertices, including two

revisits to the same vertex, such that there are no immediate revisits to any vertex.

The number of patrol patterns that are generated for a particular number of revisits is based on the number of vertices n in the graph. For the case of two revisits in the SPR2 method, there are an additional $n(n-2)((n-1)(n-1)+(n-3))$ patrol patterns to consider, which for a problem with $n = 5$ vertices is an additional 270 patrol patterns. The SPR3 method follows a similar process by conducting revisits to all combinations of three vertices such that there are no immediate revisits to any vertex. The length of the patrol patterns and the size of the sets that are generated in each of these methods are summarized in Table 3.1.

Table 3.1: Shortest path patrol pattern sets

Path generation method	Length	Number of patterns
Shortest path (SP)	$\leq n$	$2^n - 1$
Shortest path with one revisit (SPR1)	$n+1$	$n^2 - 2n$
Shortest path with two revisits (SPR2)	$n+2$	$n^4 - 3n^3 + 4n$
Shortest path with three revisits (SPR3)	$n+3$	$n^6 - 3n^5 - 5n^4 + 19n^3 - 20n$

A summary of representative patrol pattern sizes for the type of problems that we consider is presented in Table 3.2. As revisits are increased to four and beyond, there are very large increases in the number of patrol patterns without much further improvement in performance.

Table 3.2: Example numbers of shortest-path patrol patterns

Path	$n=5$	$n=6$	$n=7$	$n=8$	$n=9$	$n=10$	$n=11$	$n=12$
SP	31	63	127	255	511	1,023	2,047	4,095
SPR1	15	24	35	48	63	80	99	120
SPR2	270	672	1,400	2,592	4,410	7,040	10,692	15,600
SPR3	5,400	20,832	61,600	152,928	335,160	668,800	1,240,272	2,168,400

Patrol Patterns Based on Fictitious Play

We consider an additional group of patrol patterns that are generated using fictitious play as described by Robinson (1951). She shows that an iterative method can be used to generate mixed strategies in a two-person zero-sum game that will converge to the optimal solution. In this iterative method of play, each player arbitrarily chooses a pure

strategy in the first round. In subsequent rounds, each player chooses a pure strategy that will produce the best expected value against the mixture of strategies used by the other player in all the previous rounds.

We compute the attacker's mixed strategy $(p_1, \ldots, p_n)$ based on the mixture of strategies used by the patroller in the previous rounds. Based on that probability vector, we can use the IHT and IHE heuristic methods from the random-attacker case presented in Chapter 2 to generate a new patrol pattern for the patroller. The following algorithm is adapted from Lin et al. (2013):

1. In round 1, each player picks a strategy.
   (a) Denote by $\xi^{(d)}$ the patrol pattern used by the patroller in round d. Choose $\xi^{(1)}$ to be the shortest-path patrol pattern.
   (b) Let the attacker pick the vertex j that has the highest cost in the shortest-path patrol to attack. Use r_i, for $i \in N$, to keep track of the number of times vertex i is picked by the attacker. Initialize $r_j = 1$ and $r_i = 0$, for $i \in N, i \neq j$.
2. Repeat the following steps for the predetermined number of rounds, ν. In round $d \geq 2$,
   (a) Set $p_i = r_i / \sum_{k=1}^{n} r_k$, which represents the attacker's mixed strategy based on his attack history from rounds 1 to $d-1$. Use the random-attacker heuristic method to generate a patrol pattern $\xi^{(d)}$.
   (b) Find the best vertex for the attacker to attack by assuming the patroller uses patrol pattern $\xi^{(j)}, j = 1, \ldots, (m-1)$, each with probability $1/(m-1)$. If attacking vertex i yields the highest expected cost, set $r_i \leftarrow r_i + 1$.

Thus, we can generate two groups of patrol patterns for use in the strategic-attacker heuristic method: the shortest-path patrol and its associated derived patrol patterns, and a set of patrol patterns determined by an iterative method using fictitious play. The heuristic method in the case of fictitious play will have two parameters, the set L of look-ahead depth parameters to be used with the IHT and IHE methods, and the number of iterations of fictitious play, ν.

For a graph with n vertices, we generate $2^n - 1 + n(n-2)$ patrol patterns in the first group when using the SP and SPR1 patrol pattern sets. In the second group we generate up to $|L| \times \nu$ patrol patterns. The actual number of patrol patterns considered in the problem is often much smaller than $[2^n + n^2 - 2n - 1] + [|L| \times \nu]$, since many of the patrol

patterns generated during the fictitious-play algorithm will be identical or will produce identical performance.

3.4 Lower Bound

When the optimal solution cannot be determined due to the size of a problem, it is valuable to have a way to evaluate a heuristic solution. For this purpose, we provide a method to compute a lower bound for the optimal solution in the strategic-attacker problem. This is a modification of the discrete-time method presented in Lin et al. (2013) for our continuous-time problem.

To determine a lower bound for the optimal solution, we formulate a linear program. We define y_{ir} as the rate at which an inspection is completed at vertex i, with the last inspection at that vertex having been completed exactly r time units ago.

For example, consider a patrol pattern of total length $\tau = 17$ where inspections are completed at vertex 1 at times $2 - 5 - 7 - 10 - 14 - 17$. The times between inspections are $2 - 3 - 2 - 3 - 4 - 3$. The inspection rates at vertex 1 using this patrol pattern are $y_{12} = 2/17, y_{13} = 3/17$, and $y_{14} = 1/17$. It follows that there is a total inspection rate constraint for any vertex i that is inspected during a patrol pattern:

$$\sum_{r=1}^{\infty} y_{ir} r = 1.$$

If a vertex is not visited at all during a patrol pattern, then the total inspection rate at that vertex will be 0. Therefore, in order to create a total-rate constraint for all vertices and all patrol policies, we use

$$\sum_{r=1}^{\infty} y_{ir} r \leq 1, \qquad \forall\, i \in N. \tag{3.2}$$

Since we consider this problem in continuous time, we must modify the definition of the inspection rate in order to use it as a variable in a linear program. Recall that the attack time at vertex i is bounded by B_i. We divide the time interval $[0, B_i]$ at vertex i into m equal length subintervals. We then define an inspection rate y_{iq}, for $q = 1, \ldots, (m-1)$, as the rate at which vertex i is inspected with the previous inspection having been completed

at time in $\left[\frac{(q-1)B_i}{m}, \frac{qB_i}{m}\right)$, and y_{im} as the rate at which vertex i is inspected with the previous inspection having been completed at least $(\frac{m-1}{m})B_i$ time units ago.

Again consider the example of a patrol pattern of total length $\tau = 17$ where inspections are completed at vertex 1 at times $2 - 5 - 7 - 10 - 14 - 17$. Suppose that $B_1 = 9.6$ and we choose $m = 8$. Table 3.3 indicates the number of inspections that are completed in each time interval.

Table 3.3: Example case of time-interval inspections.

q	Interval	Inspections
1	[0, 1.2)	0
2	[1.2, 2.4)	2
3	[2.4, 3.6)	3
4	[3.6, 4.8)	1
5	[4.8, 6.0)	0
6	[6.0, 7.2)	0
7	[7.2, 8.4)	0
8	[8.4, ∞)	0

Thus, the inspection rates y_{iq} at vertex $i = 1$ for this patrol pattern are $y_{12} = 2/17, y_{13} = 3/17, y_{14} = 1/17$, and $y_{11} = y_{15} = y_{16} = y_{17} = y_{18} = 0$.

Since the inspection times are broken into m discrete time intervals, the identity in (3.2) becomes

$$\sum_{q=1}^{m} y_{iq} \frac{(q-1)B_i}{m} \leq 1, \qquad \forall\, i \in N.$$

We now focus on a single vertex in order to quantify the long-run cost at that vertex. Define $R_i(t)$ as the expected cost that can be avoided for completing an inspection at vertex i if the previous inspection was completed t time units ago. This is equivalent to the expected number of ongoing attacks at vertex i at time t multiplied by c_i, so

$$R_i(t) = c_i \lambda_i \int_0^t P(X_i > s)\, ds\,.$$

We also define

$$R_{iq} = R_i\left(\frac{qB_i}{m}\right), \qquad q = 1, \ldots, m,$$

as the cost that can be avoided at vertex i for completing an inspection at time $q(B_i/m)$.

Although we do not know the exact value of the expected cost at vertex i, we do know that

$$\left(c_i - \frac{1}{\lambda_i}\sum_{q=1}^{m} y_{iq}R_{iq}\right) \leq [\text{expected cost at vertex } i] \leq \left(c_i - \frac{1}{\lambda_i}\sum_{q=1}^{m} y_{iq}R_{i(q-1)}\right).$$

Therefore, the expected cost incurred at vertex i will be at least

$$c_i - \frac{1}{\lambda_i}\sum_{q=1}^{m} y_{iq}R_{iq}, \qquad \forall\, i \in N, \tag{3.3}$$

because the expression in (3.3) will take credit for avoiding cost in the entire interval $\left[0, \frac{qB_i}{m}\right)$ at the constant value represented by $R_i(\frac{qB_i}{m})$ times the inspection rate y_{iq}. Thus, the value in (3.3) represents a lower bound for the expected cost for each attack at vertex i.

To formulate a linear program to determine a lower bound for the optimal solution, we also incorporate constraints that account for graph structure. Define x_{ij} as the rate at which a patroller travels from vertex i to vertex j and conducts an inspection at vertex j, for $i, j \in N$. Recall that t_{ij} represents the time required for a patroller to travel from vertex i to vertex j and conduct an inspection at vertex j. On a graph with a single patroller, the following total-rate constraint applies:

$$\sum_{i,j \in N} x_{ij}t_{ij} = 1. \tag{3.4}$$

Since the total rate of arrivals to a vertex must equal the total rate of departures from a vertex, we also observe that

$$\sum_{j \in N} x_{ij} = \sum_{j \in N} x_{ji}, \qquad \forall\, i \in N.$$

The variables x_{ij} and y_{iq} are connected through the equation

$$\sum_{q=1}^{m} y_{iq} = \sum_{j \in N} x_{ij}, \qquad \forall\, i \in N,$$

since both sides represent the long-run inspection rate at vertex i.

We now formulate a linear program to determine the lower bound for the optimal solution in the single patroller against strategic attackers problem, which we refer to as the lower bound linear program (LBLP):

$$\min_{x,y} z^{\mathrm{LB}} \tag{3.5a}$$

$$\text{subject to} \quad c_i - \frac{1}{\lambda_i} \sum_{q=1}^{m} y_{iq} R_{iq} \leq z^{\mathrm{LB}}, \qquad \forall\, i \in N, \tag{3.5b}$$

$$\sum_{q=1}^{m} y_{iq} \frac{(q-1)B_i}{m} \leq 1, \qquad \forall\, i \in N, \tag{3.5c}$$

$$\sum_{j \in N} x_{ij} - \sum_{j \in N} x_{ji} = 0, \qquad \forall\, i \in N, \tag{3.5d}$$

$$\sum_{q=1}^{m} y_{iq} - \sum_{j \in N} x_{ji} = 0, \qquad \forall\, i \in N, \tag{3.5e}$$

$$\sum_{i,j \in N} x_{ij} t_{ij} = 1, \tag{3.5f}$$

$$x_{ij} \geq 0, \qquad \forall\, i, j \in N, \tag{3.5g}$$

$$y_{iq} \geq 0, \qquad \forall\, i \in N; q = 1, \ldots, m. \tag{3.5h}$$

The decision variables in this problem are x_{ij}, the rate that the patroller transits from vertex i to vertex j; and y_{iq}, the rate that an inspection is completed at vertex i with the time since the last inspection falling in $\left[\frac{(q-1)B_i}{m}, \frac{qB_i}{m}\right)$.

In this linear program, we seek to minimize the maximum expected cost for each attack across all n vertices, which is ensured by constraint (3.5b). We observe the total inspection rate constraints at each vertex with (3.5c). We also observe the network balance of flow and total arrival and inspection rate equality constraints in (3.5d) and (3.5e). Finally, we observe the total transit rate constraint on a single patroller in (3.5f), and the non-negativity constraint on patroller transit rates and inspection rates in (3.5g) and (3.5h).

While the preceding linear program will produce a valid lower bound, it can be quite loose. We add additional constraints to the linear program in order to tighten the lower bound by limiting the rate of reinspections at a vertex and by considering the transit time that is required between vertices.

To account for the action of a patroller electing to stay at a vertex to conduct an additional inspection, define

$$a_i = \left\lceil \frac{v_i}{(B_i/m)} \right\rceil, \qquad \forall\, i \in N,$$

as the number of subintervals needed for the patroller to inspect vertex i again without leaving vertex i; and require that

$$\sum_{q=1}^{a_i} y_{iq} \geq x_{ii}, \qquad \forall\, i \in N, \tag{3.6}$$

which ensures the total rate of inspections at vertex i in the time interval it takes to conduct an inspection is at least equal to the rate of reinspections at vertex i.

We also add constraints to the linear program to account for the patroller's transit rate from vertex i to j and back to vertex i, denoted by u_{iji}, for $i \neq j$, as follows:

$$u_{iji} \leq x_{ij}, \qquad \forall\, i, j \in N; i \neq j, \tag{3.7a}$$

$$u_{iji} \leq x_{ji}, \qquad \forall\, i, j \in N; i \neq j, \tag{3.7b}$$

$$x_{ij} - \sum_{k \neq i} x_{jk} \leq u_{iji}, \qquad \forall\, i, j \in N; i \neq j. \tag{3.7c}$$

Since the rate that a patroller transits from vertex i to j must be at least equal to the rate that the patroller transits from vertex i to j and back to vertex i, we include constraint (3.7a). The same reasoning applies to constraint (3.7b). We also observe in (3.7c) that the rate the patroller transits from vertex i to j and back to vertex i must be at least equal to the rate that he transits from vertex i to j, minus the rate he transits from vertex j to any vertex other than i.

It also holds that the inspection rate at vertex i must be at least equal to the rate that the patroller transits from vertex i to j and back to vertex i. To incorporate this constraint, define

$$g_{iji} = \left\lceil \frac{t_{ij} + t_{ji}}{(B_i/m)} \right\rceil, \qquad \forall\, i, j \in N; i \neq j,$$

and require that

$$\sum_{q=1}^{g_{iji}} y_{iq} \geq x_{ii} + u_{iji}, \qquad \forall\, i, j \in N; i \neq j, \tag{3.8}$$

where x_{ii} is the rate that the patroller remains at vertex i to conduct an additional inspection and u_{iji} is the rate that the patroller transits from vertex i to j and back to vertex i.

We can continue this same idea to account for paths that visit at least two vertices prior to returning to vertex i and define w_{ijki} as the rate at which the patroller transits from vertex i to vertex j to vertex k and returns immediately to vertex i. Based on the patroller's transit rate from vertex i to j to k and back to vertex i, for $i \neq j, k$, we add the following additional constraints to the linear program:

$$w_{ijki} \leq x_{ij}, \qquad \forall\, i, j, k \in N; i \neq j, k, \tag{3.9a}$$

$$w_{ijki} \leq x_{jk}, \qquad \forall\, i, j, k \in N; i \neq j, k, \tag{3.9b}$$

$$w_{ijki} \leq x_{ki}, \qquad \forall\, i, j, k \in N; i \neq j, k, \tag{3.9c}$$

$$x_{ij} - \sum_{l \neq k} x_{jl} - \sum_{l \neq i} x_{kl} \leq w_{ijki}, \qquad \forall\, i, j, k \in N; i \neq j, k. \tag{3.9d}$$

Since the rate that a patroller transits from vertex i to j must be at least equal to the rate that the patroller transits from vertex i to j to k and back to vertex i, we include constraint (3.9a). The same reasoning applies to constraints (3.9c) and (3.9d). We also observe in (3.9d) that the rate the patroller transits from vertex i to j to k and back to vertex i must be at least equal to the rate that he transits from vertex i to j, minus the rate he transits from vertex j to any vertex other than k and the rate he transits from vertex k to any vertex other than i.

It also holds that the inspection rate at vertex i must be at least equal to the rate that the patroller transits from vertex i to j to k and back to vertex i. To incorporate this constraint, define

$$h_{ijki} = \left\lceil \frac{t_{ij} + t_{jk} + t_{ki}}{(B_i/m)} \right\rceil, \qquad \forall\, i, j, k \in N; i \neq j, k,$$

and require that

$$\sum_{q=1}^{h_{ijki}} y_{iq} \geq x_{ii} + u_{iji} + w_{ijki}, \qquad \forall\, i, j, k \in N; i \neq j, k, \tag{3.10}$$

where x_{ii} is the rate that the patroller remains at vertex i to conduct an additional

inspection; u_{iji} is the rate that the patroller transits from vertex i to j and back to vertex i; and w_{ijki} is the rate that the patroller transits from vertex i to j to k and then back to vertex i.

We add constraints (3.6), (3.7a), (3.7b), (3.7c), (3.8), (3.9a), (3.9b), (3.9c), (3.9d), and (3.10) to the LBLP, which considerably tightens the lower bound. We could continue this same idea to account for paths that visit three or more vertices before returning to a starting vertex; however, for the size of the graphs that we consider, that would involve many more variables with negligible gains in performance. The number of decision variables in this linear program is $n^2 + mn$. The number of constraints is $5n^3 + 5n^2 + (m-10)n + 1$. For a problem with $n = 5$ and $m = 100$, there are 525 decision variables and 1,201 constraints. In our numerical experiments, it takes on average 0.61 second to compute a lower bound for a problem of this size.

3.5 Numerical Experiments

To test the shortest-path and fictitious-play (FP) heuristic methods, we conduct several numerical experiments. We compare the results obtained from using the heuristic methods to the optimal solution. We also report the computation time required. Additionally, we compute a lower bound for the optimal solution using the linear program described in Section 3.4. Based on these results, we make conclusions on the efficacy of the heuristics, as well as make recommendations for the best use of the shortest-path and fictitious-play methods.

We test the same five problem cases for strategic attackers that we did for random attackers in Chapter 2. In each case, we use the same 1,000 problem scenarios that were randomly generated for the random-attacker experiments. The attack probability vector is omitted for the strategic-attacker problems, but all other data remain the same. We conduct our baseline experiments on a graph with $n = 5$ vertices.

In our experimental results, the optimal solution that is obtained from using the SALP is indicated by z^{OPT}. The lower bound that is obtained from using the LBLP is indicated by z^{LB}. Solutions obtained from using a heuristic method are indicated by z^{H}, where H indicates the heuristic method that was used.

3.5.1 Baseline Problems

For our baseline problem, we consider the case where a patroller spends about half of the time traveling and half of the time inspecting vertices. We determine the optimal solution using the SALP from Section 3.2 and a solution using the heuristic methods from Section 3.3. The SALP on average uses 5,920 decision variables and 7,110 constraints for a problem size with 1,184 states. The optimal solution takes on average 20.68 seconds to compute. We compare the solution obtained from the heuristic method to the optimal solution. We also determine a lower bound for the optimal solution using the LBLP in Section 3.4, and compare that result to the optimal solution.

Using 1,000 problem instances, we test the shortest-path method with the SP, SPR1, SPR2, and SPR3 patrol pattern sets. We also test the FP method with 10, 20, 30, and 50 iterations. Results of the baseline experiments are presented in Table 3.4. Excellent performance is observed with both the shortest-path SPR2 and SPR3 methods and the FP method with 50 iterations. Each of these methods returns a solution within 1.11 percent of the optimal solution in at least 90 percent of the problem instances. The shortest-path method uses considerably less computation time than the FP method in all cases. A tight lower bound for the optimal solution was also obtained, with an average difference between the lower bound and the optimal solution of 1.20 percent.

Table 3.4: Performance of the shortest-path and fictitious-play heuristic methods on a complete graph with $n = 5$ vertices, based on 1,000 randomly generated problem instances with average inspection times that are comparable to average travel times. Mean, 50th, 75th and 90th percentile performance is indicated as the percentage excess over the optimal solution. The lower bound is reported as $(z^{\mathrm{LB}} - z^{\mathrm{OPT}})/z^{\mathrm{OPT}}$ in percentage.

Heuristic method	Percent over optimum				Time
	Mean	50th	75th	90th	(sec)
Shortest-path (SP)	1.95	1.18	2.53	4.45	< 0.01
SP with one revisit (SPR1)	0.72	0.39	0.93	1.82	0.04
SP with two revisits (SPR2)	0.39	0.12	0.47	1.11	0.52
SP with three revisits (SPR3)	0.28	0.05	0.28	0.80	6.15
Fictitious play ($\nu = 10$)	3.76	3.11	5.23	8.18	85.51
Fictitious play ($\nu = 20$)	1.85	1.39	2.42	4.13	167.56
Fictitious play ($\nu = 30$)	0.79	0.45	0.90	2.11	255.45
Fictitious play ($\nu = 50$)	0.32	0.22	0.43	0.73	425.45
Lower bound	-1.20	-0.29	-1.17	-3.35	0.61

We also test combinations of the two-person zero-sum game matrices that are produced from each heuristic method. When the game matrices are combined, the resulting performance can be no worse than what is obtained with each of the individual methods since additional patrol patterns are being considered. The mean and 90th percentile performance results are presented in Table 3.5. We see an improvement in performance when the methods are combined, but it is generally not significant enough to justify the additional computation time required by the FP method. It requires at least 20 iterations of FP combined with the SPR2 set and at least 30 iterations of FP combined with the SPR1 set to improve upon the performance obtained from using the SPR3 patrol pattern set alone.

Table 3.5: Mean and (90th percentile) performance of the shortest-path and fictitious-play heuristic methods on a complete graph with $n = 5$ vertices, based on 1,000 randomly generated problem instances with average inspection times that are comparable to average travel times, reported as the percentage excess over the optimal solution.

	Percent over optimum					Time
FP/SP	—	SP	SPR1	SPR2	SPR3	(sec)
—	—	1.95 (4.45)	0.72 (1.82)	0.39 (1.11)	0.28 (0.80)	
FP 10	3.76 (8.18)	1.70 (3.98)	0.57 (1.75)	0.32 (1.04)	0.23 (0.72)	85.51
FP 20	1.85 (4.13)	0.99 (2.43)	0.36 (1.16)	0.19 (0.66)	0.16 (0.42)	167.56
FP 30	0.79 (2.11)	0.50 (1.40)	0.24 (0.69)	0.13 (0.43)	0.10 (0.27)	255.45
FP 50	0.32 (0.73)	0.26 (0.67)	0.13 (0.43)	0.11 (0.28)	0.08 (0.17)	425.45
Time (sec)		< 0.01	0.04	0.52	6.15	

3.5.2 Recommendations Based on Numerical Experiments

We see very favorable results with the SP method. In at least 90 percent of the problem instances, we observe results within 1.11 percent of the optimal solution when using the SPR2 method and within 0.80 percent of the optimal solution when using the SPR3 method. For problems with $n = 5$, the SPR2 method required 0.52 second on average and the SPR3 method required 6.15 seconds on average to return a solution. The advantage to the SP method is that it provides excellent results for very little computation time.

We can generate additional effective patrol patterns for consideration in determining a randomized patrol policy, and further refine the overall solution, by considering the patterns obtained from multiple iterations of FP. The solution improves as the number of iterations of FP increases, but comes at a cost of significantly increased computation

time. In at least 90 percent of problem instances, we see solutions within 2.11 percent of optimal when using 30 iterations of FP and within 0.73 percent of optimal when using 50 iterations of FP. These problem instances required on average 4.25 minutes and 7 minutes, respectively, to return a solution.

Based on the experimental results, we recommend using the SPR2 method for the strategic-attacker problem. The use of the FP method is not recommend in most situations due to the high amount of computation time required.

3.5.3 Performance on Smaller and Larger Graphs

In addition to problems with $n = 5$, we test the heuristic methods on smaller and larger size graphs. For graphs with $n = 3$, 4, and 5, we compare the performance of the SPR2 heuristic to the optimal solution. Results are presented in Table 3.6.

Table 3.6: Performance of the SPR2 shortest-path heuristic on a complete graph, based on 1,000 randomly generated problem instances with average inspection times comparable to average travel times. Mean, 50th, 75th and 90th percentile performance is indicated as the percentage over the optimum solution. The mean lower bound is reported as $(z^{\mathrm{LB}} - z^{\mathrm{OPT}})/z^{\mathrm{OPT}}$ in percentage.

Vertices	Percent over optimum				Time (sec)		Lower
(n)	Mean	50th	75th	90th	z^{SPR2}	z^{OPT}	bound
3	0.00	0.00	0.00	0.00	0.03	< 0.01	0.00
4	0.10	0.00	0.04	0.17	0.08	0.23	-0.04
5	0.39	0.12	0.47	1.11	0.52	20.68	-1.27

We note that the SPR2 heuristic method works extremely well for graphs smaller than $n = 5$, returning a solution that is within 0.17 percent of optimal in 90 percent of the problem instances with computation times of less than 0.1 second. For graphs with $n =$ 6, 7, 8, and 9, we compare the performance of the heuristic to the lower bound. Results are presented in Table 3.7. We use the lower bound for a comparison because, in our experiments, it is not practical to compute the optimal solution for graphs with $n > 5$ due to computer memory limitations.

We note that the SPR2 shortest-path heuristic method returns results that are within 10 percent of the lower bound in 90 percent of the problem instances for $n = 6$, and within 16 percent of the lower bound in 90 percent of problem instances for $n = 9$. These solutions take on average 0.58 second and 7.98 seconds, respectively, to compute.

Table 3.7: Performance of the SPR2 shortest-path heuristic on a complete graph, based on 1,000 randomly generated problem scenarios with average inspection times that are comparable to average travel times. Mean, 50th, 75th and 90th percentile performance is indicated as the percentage excess above the lower bound, reported as $(z^{\text{SPR2}} - z^{\text{LB}})/z^{\text{LB}}$ in percentage.

Vertices	Percent over lower bound				Time
(n)	Mean	50th	75th	90th	(sec)
3	0.00	0.00	0.00	0.00	0.03
4	0.14	0.03	0.08	0.22	0.08
5	1.66	0.75	1.57	3.15	0.52
6	3.58	2.03	4.63	9.71	0.58
7	4.93	3.03	5.75	11.98	1.35
8	5.84	4.54	8.64	12.47	3.34
9	7.56	5.67	10.49	15.93	7.98

3.5.4 Performance on Additional Graph Structures

In addition to problems on a complete graph, we test the SPR2 heuristic method on several additional graph structures. Specifically, we consider line graphs, circle graphs, and random trees. We use the procedures from Section 2.4.1 to generate 1,000 random problem instances for problem cases with $n = 4$, 5, 6, and 7 vertices.

To construct a line graph, we randomly assign $n - 1$ edges between n vertices, such that the degree of each vertex is at least one but no more than two. To construct a circle graph, we randomly assign n edges between n vertices, such that the degree of each vertex is exactly two. To construct a random tree, we randomly assign $n - 1$ edges between n vertices, such that the degree of each vertex is at least one and there is at least one vertex of degree greater than two, which excludes line graphs from the random tree category.

We still allow a patroller to travel between any two vertices in order to determine a patrol policy. For these additional graph structures, a patroller may have to travel through one or more interim vertices (without conducting inspections at those vertices) in order to arrive at the destination vertex.

We consider cases where average travel times are comparable to average inspection times. To do this, we scale the travel times between each pair of vertices based on the graph structure. Specifically for any particular graph, we determine the average number of edges between each pair of vertices and divide the travel times by that average value. This

produces average total travel times between each pair of vertices that are comparable to average inspection times. We construct a distance matrix D using these scaled travel times. The distance d_{ij} is the total travel time along the shortest path in the graph between each pair of vertices i and j, for $i, j \in N$.

Results for these additional graph structures with $n = 4$, 5, 6, and 7 are presented in Table 3.8. For graphs with $n \leq 5$, we compare the performance of the heuristic to the optimal solution as well as to the lower bound. For graphs with $n \geq 6$, we compare the heuristic to the lower bound, since an optimal solution cannot be determined for problems of this size.

Table 3.8: Mean performance of the SPR2 heuristic method on additional graph structures, based on 1,000 randomly generated problem scenarios for average inspection times that are comparable to average travel times. Performance is indicated as the mean percentage over optimum for problems where an optimal solution can be determined using the SALP, and the mean percentage over lower bound for all problems.

Graph	Vertices	Performance (%)		Time (sec)	
	(n)	$z^{\text{SPR2}}/z^{\text{OPT}}$	$z^{\text{SPR2}}/z^{\text{LB}}$	z^{SPR2}	z^{OPT}
Complete	4	0.10	0.12	0.08	0.23
Complete	5	0.39	1.66	0.52	20.68
Complete	6	—	3.58	0.58	—
Complete	7	—	4.93	1.35	—
Line	4	0.08	0.10	0.09	0.28
Line	5	0.26	0.90	0.46	35.84
Line	6	—	8.11	0.53	—
Line	7	—	11.12	1.31	—
Circle	4	0.12	0.15	0.08	0.29
Circle	5	0.50	1.18	0.50	22.25
Circle	6	—	2.32	0.54	—
Circle	7	—	3.73	1.29	—
Random tree	4	0.05	0.14	0.09	0.23
Random tree	5	0.15	0.84	0.52	28.62
Random tree	6	—	4.79	0.55	—
Random tree	7	—	5.99	1.35	—

These results indicate that the shortest-path heuristic method can be used very effectively for the strategic-attacker problem on several different graph structures and sizes. For problems with $n = 5$, where an optimal solution can be determined, the SPR2 method returns a solution on average that is within 0.50 percent of optimal. These solutions

take approximately 0.5 second to compute, which is 40 times less than the time required to compute the optimal solution. For problems with $n = 7$, where an optimal solution cannot be determined, the heuristic produces on average a result within 3.73 percent of the lower bound on a circle graph, and within 11.12 percent of the lower bound on a line graph. These solutions take less than 1.5 seconds to compute.

3.5.5 Sensitivity Analysis

In addition to the baseline problems, we consider the case where a patroller needs to spend more time conducting inspections than he does traveling between vertices; and the case where the patroller needs to spend more time traveling between vertices than he does conducting inspections. The five specific cases we consider in the numerical experiments are summarized in Table 3.9. Case III generated the smallest number of states and had the highest long-run cost on average. It also generated the tightest lower bound for the optimal solution. Case IV generated the largest number of states and had the lowest long-run cost on average. It also generated the loosest lower bound for the optimal solution.

Table 3.9: Summary of numerical experiments for strategic attackers. The mean lower bound is reported as $(z^{\mathrm{LB}} - z^{\mathrm{OPT}})/z^{\mathrm{OPT}}$ in percentage.

Parameter	Case I	Case II	Case III	Case IV	Case V
Travel time	1×	1×	1×	2×	2×
Inspection time	1×	2×	2×	1×	1×
Attack time	1×	1.5×	1×	1.5×	1×
Mean number of states, $\lvert\Omega\rvert$	1,184	633	102	3,938	318
Mean number of decision variables	5,920	3,165	510	19,690	1,590
Mean number of constraints	7,110	3,804	613	23,679	1,914
Mean optimal long-run cost	0.4892	0.5085	0.6589	0.4761	0.6224
Mean optimal computation time (sec)	20.68	4.99	0.11	574.85	2.11
Lower bound	−1.20	−0.20	−0.03	−4.81	−0.88

The mean performance results for problem cases II through V using both the SP and FP methods are presented in Table 3.10. The 90th percentile performance results are presented in Table 3.11. In each of the problem cases, very favorable results are obtained using the SP heuristic method. In at least 90 percent of the problem instances, the SPR2 method returns a solution within 1.51 percent of optimal. These solutions take 0.52 second to compute on average.

Table 3.10: Mean performance of the shortest-path and fictitious-play methods, based on 1,000 randomly generated problem scenarios for each case. Performance is indicated as the percentage excess over the optimal solution. Shortest-path computation time is indicated for the SPR2 heuristic.

Case		Percent over optimum (Mean)				Time (sec)
	FP/SP	—	SP	SPR2	SPR3	
II	—	—	1.26	0.21	0.14	0.52
	FP 10	3.32	1.23	0.18	0.12	29.71
	FP 20	1.20	0.67	0.13	0.10	59.52
	FP 30	0.60	0.44	0.10	0.07	89.92
	FP 50	0.30	0.27	0.08	0.04	151.99
III	—	—	0.41	0.22	0.17	0.50
	FP 10	1.66	0.39	0.19	0.15	2.25
	FP 20	0.74	0.27	0.16	0.12	4.75
	FP 30	0.50	0.15	0.10	0.07	7.38
	FP 50	0.37	0.15	0.09	0.05	12.79
IV	—	—	2.65	0.50	0.34	0.50
	FP 10	4.49	2.15	0.34	0.26	717.60
	FP 20	2.19	1.42	0.26	0.19	1,337.60
	FP 30	1.08	0.80	0.12	0.09	1,977.97
V	—	—	0.90	0.53	0.44	0.47
	FP 10	2.96	0.74	0.45	0.38	14.30
	FP 20	1.37	0.51	0.31	0.26	29.78
	FP 30	0.83	0.60	0.22	0.17	47.16
	FP 50	0.51	0.17	0.16	0.11	78.87

Table 3.11: 90th percentile performance of the shortest-path and fictitious-play methods, based on 1,000 randomly generated problem scenarios for each case. Performance is indicated as the percentage excess over the optimal solution. Shortest-path computation time is indicated for the SPR2 heuristic.

Case		Percent over optimum (90th percentile)				Time (sec)
	FP/SP	—	SP	SPR2	SPR3	
II	—	—	3.21	0.69	0.49	0.52
	FP 10	5.95	2.94	0.66	0.42	29.71
	FP 20	2.47	1.64	0.49	0.33	59.52
	FP 30	1.35	1.05	0.37	0.24	89.92
	FP 50	0.79	0.47	0.23	0.16	151.99
III	—	—	1.06	0.60	0.53	0.50
	FP 10	3.08	1.04	0.45	0.39	2.25
	FP 20	1.47	0.78	0.39	0.32	4.75
	FP 30	1.12	0.69	0.30	0.24	7.38
	FP 50	0.77	0.36	0.28	0.19	12.79
IV	—	—	5.44	1.51	1.06	0.50
	FP 10	8.63	4.58	0.87	0.76	717.60
	FP 20	4.72	3.84	0.82	0.68	1,337.60
	FP 30	2.78	2.18	0.27	0.21	1,977.97
V	—	—	1.90	1.26	1.16	0.47
	FP 10	5.79	1.77	1.02	0.85	14.30
	FP 20	3.07	1.30	0.73	0.61	29.78
	FP 30	1.94	1.27	0.60	0.49	47.16
	FP 50	1.32	0.42	0.34	0.24	78.87

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4:
Multiple Patrollers Against Strategic Attackers

In this chapter, we consider the case of multiple patrollers against strategic attackers, where an attacker will actively choose a location to attack in order to incur the highest expected cost. In Section 4.1, we introduce a patrol model where k patrollers are assigned to patrol a graph consisting of n vertices, with $k \leq n$. In Section 4.2, we present a heuristic method for determining a patrol policy based on two types of pure strategies. We present a strategy for the patrollers based on set partitions, where the patrol team divides the vertices among the individual patrollers with each patroller then executing his best strategy for patrolling the assigned subset of vertices. We also present a strategy for the patrollers based on the shortest Hamiltonian cycle in the graph, where each patroller follows the same shortest Hamiltonian cycle at evenly spaced time intervals. In Section 4.3, we present a method to compute a lower bound for the optimal solution. We conduct numerical experiments for several scenarios and present the results in Section 4.4.

4.1 Patrol Model

We introduce a patrol model where k patrollers are assigned to patrol a graph consisting of n vertices, with $k \leq n$. This problem can arise when an area of interest (AOI) is too large for a single patroller to cover effectively, either due to the total number of potential attack locations or long travel times between locations. It can also be applicable if the expected cost per attack in a problem is determined to be too large for a single patroller, perhaps due to large costs for successful attacks or short attack time distributions at one or more locations, and the assignment of additional patrollers to the problem is an option.

In our model, the attacker and patrollers play a simultaneous-move two-person zero-sum game where the attacker is trying to maximize the cost incurred due to a successful attack and the patrollers are trying to minimize it. The patrollers decide how to patrol the graph while the attacker chooses which vertex to attack.

The state space of this problem is infinite, since the time between inspections at a vertex can take on any non-negative value when there are multiple patrollers. Therefore, it is possible to determine the optimal patrol policy in only a few special cases. The objective in solving this patrol problem is to provide a feasible patrol policy for the team of patrollers

in order to keep the expected cost per attack as low as possible, which requires the use of efficient heuristics.

4.2 Heuristic Policy

In this section, we consider a heuristic method to determine a strategy for the multiple patrollers. Since for most problem instances it is impractical or impossible to consider every feasible state of the system in order to determine an optimal strategy using linear programming, as was done in Chapter 3, we instead consider a finite set of pure strategies from which the patrollers can choose. If the pure strategies in the set are effective and diverse, then we expect that the optimal mixed strategy from using only those pure strategies would produce a strong heuristic policy. The spirit of this method is the same as the heuristic in Section 3.3, where we discuss the case of single patroller against strategic attackers.

In the following sections, we consider two types of pure strategies. The first type is based on set partitions. The second type is based on the shortest-path patrol pattern that was introduced in Chapter 3. In the set-partition method, we partition the vertices into subsets with each patroller then executing the best patrol policy for his assigned subset of vertices, independent of the other patrollers. In the shortest-path patrol method, each patroller follows the same patrol cycle at evenly timed intervals so as to minimize the time between inspections at each vertex. Each of these methods will produce one or more pure strategies for the patrollers.

For each pure strategy, we compute the expected cost per attack at each of the vertices. Given a set of pure strategies, $\Theta = \{\theta_1, \theta_2, \ldots, \theta_m\}$, where each strategy has an expected cost per attack at each vertex, a two-person zero-sum game can be formulated between the attacker and the patrollers in a standard matrix form. In this game matrix, row i corresponds to the attacker choosing to attack vertex i and column j corresponds to the patrollers choosing to use strategy θ_j, for $i \in N$ and $j = 1, \ldots m$. A linear program can then be formulated to solve this two-person zero-sum matrix game (Washburn 2003). The solution to this game provides a mixed strategy for the attacker, which is a probability distribution on the vertices to attack; and a mixed strategy for the patrollers, which is a probability distribution on the set of pure strategies.

4.2.1 Pure Strategy Based on Set Partitions

One natural way for a patrol team to patrol a large graph is to divide the graph into subsets and assign each patroller to a subset of vertices. This can be done by dividing the n vertices of the graph into k mutually exclusive and exhaustive non-empty subsets. Each of the k patrollers is then assigned one of these subsets and executes his best individual patrol strategy against strategic attackers for that subset of vertices, independent of the other patrollers.

In order to assign the vertices among the k patrollers we create a *set partition*. A partition of a set N is a collection of non-empty blocks so that each element of N belongs to exactly one block (Bóna 2011). If the use of one specific partition is considered to be a pure strategy, we can develop a mixed strategy for the patrollers by allowing them to choose among several vertex set partitions.

In general, we desire that vertices within a block be close to each other in terms of distance. This reduces travel time for the individual patrollers, which can have beneficial results as previously seen in the random-attacker problem cases with shorter travel times in Chapter 2 and when using the shortest-path patrol methods against a strategic attacker in Chapter 3. We propose the following procedure as one method to create vertex set partitions in order to achieve this goal.

Determining Set Partitions

One way to determine set partitions is to consider every combination of assigning n distinct vertices to one of k patrollers, such that each patroller is assigned at least one vertex. The number of partitions of a set N, where $|N| = n$, into k non-empty blocks is denoted by the Stirling numbers of the second kind. These values are expressed as $S(n, k)$. A formula for computing Stirling numbers of the second kind is (Bóna 2011)

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n.$$

Consider as an example the combinations of vertices and patrollers that we use for the numerical experiments in this chapter. If we were to consider all the ways of assigning 25 distinct vertices among 5 patrollers, such that each patroller is assigned at least one vertex, then that number of partitions would exceed 2.4 quintillion. For 20 vertices and 4 patrollers the number of partitions is over 45 billion. For 15 vertices and 3 patrollers

the number of partitions is over 2.3 million. For the case of 10 vertices assigned to 2 patrollers, $S(10,2) = 511$, which is a more manageable yet still formidable number. Since it is impossible or impractical to consider all possible set partitions in most problem instances, we determine an effective way to create a comprehensive subset of the complete set of vertex partitions from which the patrollers can develop an effective mixed strategy.

We determine a set of several vertex partitions using a two-step process. First, we designate a set of k seed vertices that will anchor each vertex cluster. From these initial seeds, we grow the vertex clusters by adding each of the additional $n - k$ vertices to a cluster until all n vertices have been assigned to exactly one cluster.

In step one, we determine seed vertices. There are several methods for determining seed vertices. We could randomly select k vertices as seeds, but this would not necessarily be an efficient method. Alternatively, we could exhaustively consider all $\binom{n}{k}$ combinations of k vertices, which would also be inefficient due to the amount of computational resources required to create and evaluate vertex clusters based on all of these seed combinations.

We propose a greedy method to select seed vertices based on average distance as follows: For each vertex i, make i the first element in a set of seeds S. Then add the furthest vertex in terms of average travel distance from the current elements of S. Repeat until k seeds have been included in S. This method will produce n sets of seed vertices, with each vertex being an element of at least one set. Proceed as follows for each vertex $i \in N$,

1. $S = \{i\}$: Each vertex $i \in N$ will be the anchor vertex for one set of seed vertices.
2. $l \leftarrow \arg\max_{j \notin S} \left\{ \sum_{i \in S} d_{ij} / |S| \right\}$: Find the furthest vertex l in terms of average travel distance from the current vertices in the set of seeds.
3. $S \leftarrow \{l\} \cup S$: Add vertex l to the set of seeds.
4. If $|S| = k$, stop and return S; otherwise go to (2). Continue until the set of seeds has k elements.

In step two, we grow vertex clusters. There are several methods for growing clusters from the seed vertices. We could randomly assign the $n - k$ remaining vertices to the seed vertices, but again this would not necessarily be an efficient method. Alternatively, we could consider all $S(n-k, k)$ combinations of assigning the $n - k$ remaining vertices to the seed vertices, such that each cluster consists of at least two vertices. However, like the exhaustive enumeration of seed vertex combinations, this method would also be inefficient

due to the amount of computational resources required.

We propose a method for growing the vertex clusters based on average travel distance between vertices as follows: From among the remaining unassigned vertices, select the one that is closest to any cluster in terms of average travel distance to the vertices currently in each cluster, and assign that vertex to its closest cluster. Continue this process until all vertices have been assigned to a cluster. Proceed as follows for a set of seed vertices S, with $|S| = k$,

1. Relabel the k seeds to be vertices $1, 2, \ldots, k$. Let $\pi_i = \{i\}$, for $i = 1, 2, \ldots, k$, and $T = \{k+1, \ldots, n\}$. Assign each seed vertex to a distinct cluster and identify the remaining unassigned vertices.
2. Compute
$$D_{i,j} = \frac{\sum_{q \in \pi_i} d_{j,q}}{|\pi_i|}, \qquad i = 1, \ldots, k; j \in T,$$
3. Find i^* and j^* such that $D_{i^*j^*} = \min_i \min_j D_{ij}$. Update $\pi_{i^*} \leftarrow \{j^*\} \cup S_{i^*}$ and $T \leftarrow T \backslash \{j^*\}$. Add vertex j^* to its closest cluster π_{i^*} in terms of average travel distance.
4. If $T = \emptyset$, stop and return $\theta = \{\pi_1, \ldots, \pi_k\}$; otherwise go to (2). Continue until each vertex has been assigned to exactly one cluster. This algorithm will return k non-empty clusters of vertices.

Iterative Method for Improving Set Partitions

The methods described above use the travel distance between vertices when determining set partitions. Shortening the time spent on moving between vertices will generally produce favorable results, as seen in the shortest-path patrol method used for a single patroller against strategic attackers in Chapter 3. However, there are additional factors that can be considered when assigning vertices to patrollers. In addition to its location relative to other vertices in the graph, each vertex $i \in N$ has an inspection time, an attack time distribution, and a cost incurred due to an undetected attack. These parameters can help determine both the difficulty and the value of an attack against a particular vertex, and will factor into the expected cost at each vertex for any patrol policy.

For example, a vertex with a higher cost would be more attractive to an attacker for obvious reasons. Similarly a vertex with a high inspection time would be attractive to an attacker, since he would have a better chance of completing an attack before a patroller

can complete an inspection. Conversely, a vertex with a higher expected attack time would be less attractive to an attacker, due to the higher likelihood of an attack being detected. We present a method to improve the initial set of vertex partitions that was created using distances by also considering the expected cost incurred at each vertex due to an undetected attack. This method provides a way to balance the workload among the individual patrollers.

We propose a one-step policy-improvement procedure to create additional set partitions as follows. For each partition in the initial set of vertex partitions, determine the expected cost at each vertex using the methods from Chapter 3 for a single patroller against strategic attackers. Then, reassign the vertices by removing one vertex from the highest cost cluster and adding one vertex to the lowest cost cluster in order to form a new partition. To do this, we identify the vertex outside of the lowest cost cluster that is closest in terms of average travel distance to the vertices currently in the lowest cost cluster. That vertex is reassigned to the lowest cost cluster. If that vertex came from the highest cost cluster, the process terminates and returns the newly created partition. If the vertex was not removed from the highest cost cluster, then the cluster that lost that vertex must gain a replacement vertex. In this case, the process repeats with the cluster that just lost a vertex taking the closest vertex from any of the remaining clusters that have not yet gained an additional vertex. The process continues until the highest cost cluster has lost a vertex. Since there are k clusters in every partition, this process will always terminate in a maximum of k iterations.

An example of the improvement iteration procedure is shown in Figures 4.1, 4.2, and 4.3. Figure 4.1 shows an initial set partition with the expected costs for each cluster. Figure 4.2 shows the new partition that is created after one iteration of the improvement algorithm. In this case, vertices $\{5\}$ and $\{8\}$ need to be reassigned in order for the lowest cost partition to gain one vertex and the highest cost partition to lose one vertex. Expected costs are then determined for each cluster in this new partition. Figure 4.3 shows a final iteration of the improvement algorithm. In this case, vertex $\{5\}$ is reassigned in order for the lowest cost partition to gain one vertex and the highest cost partition to lose one vertex. If the iteration method is repeated for the partition in Figure 4.3, the partition in Figure 4.2 is recreated, and the improvement iteration method terminates since all new partitions based on the initial set partition presented in Figure 4.1 have been determined.

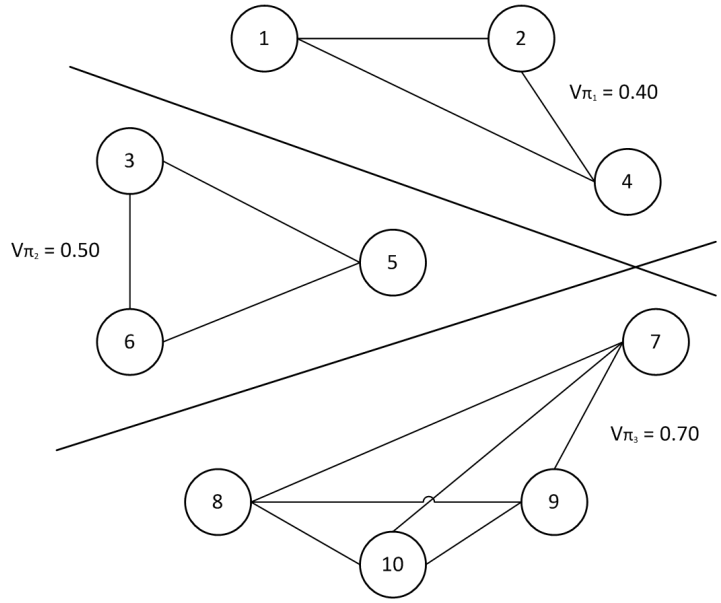

Figure 4.1: Example of a vertex set partition for $n = 10$ and $k = 3$. Cluster $\pi_3 = \{7, 8, 9, 10\}$ has the highest expected cost and will lose a vertex during an improvement iteration. Cluster $\pi_1 = \{1, 2, 4\}$ has the lowest expected cost and will gain a vertex during an improvement iteration.

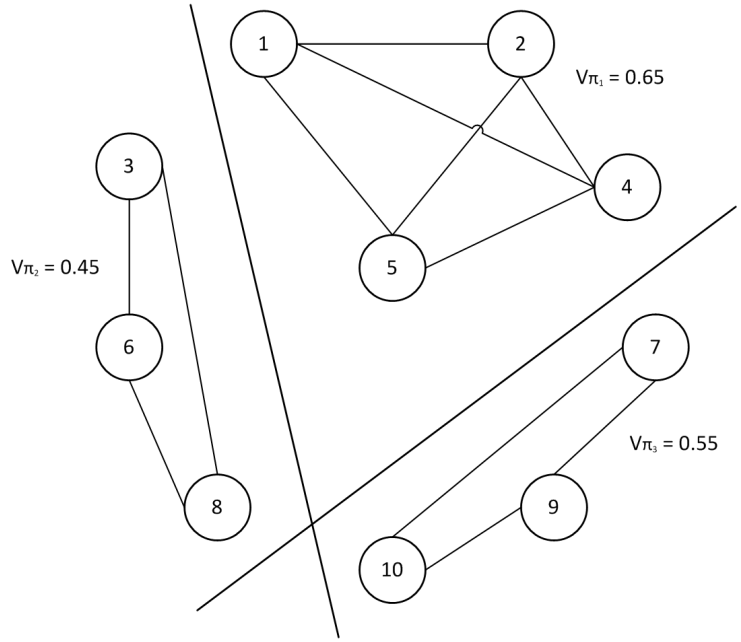

Figure 4.2: Iterated vertex set partition for $n = 10$ and $k = 3$. Cluster π_1 gained vertex 5 from cluster π_2, and cluster π_2 gained vertex 8 from cluster π_3 to complete the iteration. Now cluster $\pi_1 = \{1, 2, 4, 5\}$ has the highest expected cost and will lose a vertex in the next iteration. Cluster $\pi_2 = \{3, 6, 8\}$ has the lowest expected cost and will gain a vertex in the next iteration.

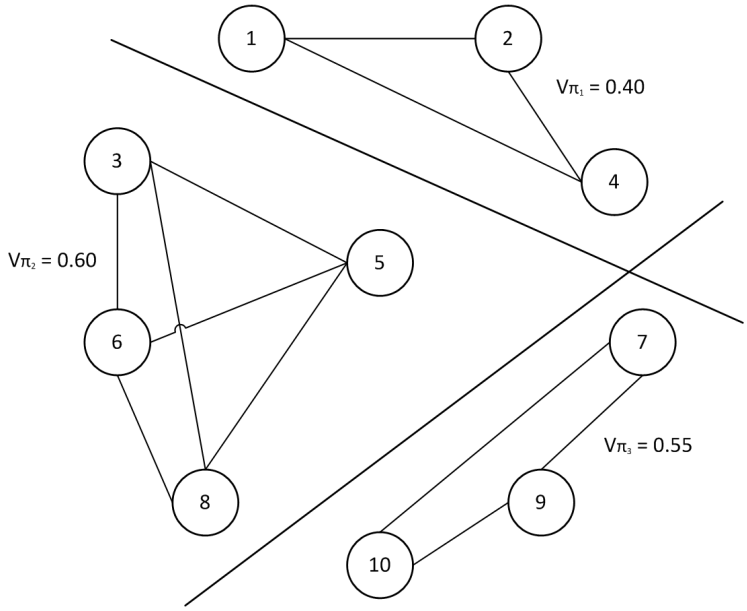

Figure 4.3: Additional iterated vertex set partition for $n = 10$ and $k = 3$. Cluster π_2 gained vertex 5 from cluster π_1 to complete the iteration.

For an initial set partition, we can apply the following algorithm to produce a new partition,

1. Relabel the vertex subsets so that π_1 is the least vulnerable (lowest cost) subset, and π_k the most vulnerable (highest cost) subset. Let $S' = \{1, \ldots, n\} \backslash \pi_1$, which represents the set of locations that are available for reassignment. Let $j \leftarrow 1$, which indicates the subset that needs an additional location.
2. Find location $l \in S'$, which is closest to π_j in terms of its average distance to all locations in π_j. Let $m \leftarrow \{i : l \in \pi_i\}$; that is, π_m is the subset that contains location l.
3. Reassign location l to subset j. That is, let $\pi_j \leftarrow \pi_j \cup \{l\}$, and $\pi_m \leftarrow \pi_m \backslash \{l\}$.
4. If $m = k$, then stop; otherwise, let $j \leftarrow m$, and $S' \leftarrow S' \backslash \{\pi_m \cup \{l\}\}$, and go to (2).

We repeat this procedure for each partition in the initial set of partitions, as well for any new partitions that are formed during the process. The process terminates when there are no new partitions to consider. This algorithm will always terminate since there are a finite number of possible vertex partitions. All partitions created using this procedure become pure strategies for the patrol team in the two-person zero-sum game between the attacker and patrollers as described in Section 4.2.

4.2.2 Pure Strategy Based on Shortest Path

We also consider a strategy based on the shortest-path patrol pattern. We do this because some vertex layouts do not allow for efficient solutions to be found using the set-partition method. One example of this is a circular layout of vertices. This situation is commonly encountered when a team of patrollers is assigned to patrol several locations along a perimeter.

An example of a circular vertex layout with 8 vertices and 2 patrollers is presented in Figure 4.4, with an arbitrary set partition, $\pi_1 = \{1, 2, 3, 4\}$ and $\pi_2 = \{5, 6, 7, 8\}$, depicted. If a circular layout of vertices was divided into clusters, each patroller would be spending a large amount of time traveling between the end vertices in a cluster for any strategy. In this case, it may be more efficient for each patroller to follow a path along the perimeter that visits every vertex, rather than be individually assigned to exclusively patrol a vertex cluster.

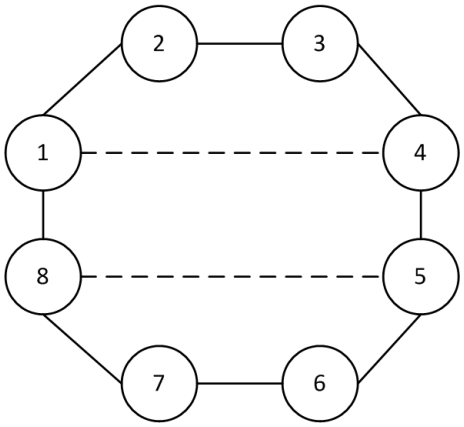

Figure 4.4: Circular vertex layout example for $n = 8$ and $k = 2$.

We determine the shortest-path patrol pattern by finding the shortest Hamiltonian cycle in the graph in terms of total travel distance, and calculate the total transit time τ that is required for a single patroller to complete this cycle. All k patrollers then follow this same patrol pattern, such that they are equally spaced and the time between patrollers at each vertex is τ/k. Recall from Section 3.3.1 that the expected cost at vertex j using this patrol pattern is

$$\rho_j = \frac{c_j k \int_0^{\tau/k} F_j(s)\, ds}{\tau}, \qquad \forall\, j \in N.$$

4.3 Lower Bound

To determine a lower bound for the optimal solution in the multiple-patroller problem, we modify the linear program that was used to compute a lower bound for the single patroller against strategic attackers problem in Section 3.4. Since we allow for multiple patrollers on a graph, we note that it is feasible for an inspection at a vertex to be completed at any time following the last inspection, including time intervals that are less than the single patroller inspection time at a vertex.

Recall the total-rate constraint on a graph with a single patroller from (3.4),

$$\sum_{i,j \in N} x_{ij} t_{ij} = 1.$$

We modify this constraint to account for the total rate that k patrollers can transit through the graph as,

$$\sum_{i,j \in N} x_{ij} t_{ij} = k, \tag{4.1}$$

and replace (3.4) with (4.1) in the linear program from Section 3.4 for use in the multiple-patroller problem.

4.4 Numerical Experiments

To test the set-partition and shortest-path strategies, we conduct several numerical experiments on a graph with n vertices and k patrollers, where $k \leq n$. We compare the results obtained from using the heuristic method to the lower bound. We also report the computation time required. Based on these results, we make conclusions on the efficacy of the heuristic method, as well as make recommendations for the best use of the set-partition and shortest-path strategies.

We test the same five problem cases for multiple patrollers as we did for a single patroller against strategic attackers in Chapter 3. We create problem scenarios using the procedures in Section 2.4.1, and generate a field of 25 vertices for use in each scenario. From this field, we randomly select the desired number of vertices for use in each problem instance. We examine cases of 10 vertices and 2 patrollers; 15 vertices and 3 patrollers; 20 vertices and 4 patrollers; and 25 vertices and 5 patrollers.

4.4.1 Baseline Problems

For the baseline problem, we consider the case where the patrollers spend about half of the time traveling and half of the time inspecting vertices. We determine a solution using the heuristic method from Section 4.2. We also determine a lower bound for the optimal solution using the linear program in Section 4.3 and compare that result to the heuristic solution.

For 1,000 problem instances, we test both the set-partition with one-step policy-improvement strategy from Section 4.2.1 and the shortest-path patrol strategy from Section 4.2.2. For the one-step policy-improvement procedure, we also note the level of improvement that was made by comparing the size and performance of the initial set of vertex partitions with the expanded set of partitions. Results are presented in Table 4.1. The greatest amount of improvement was observed on larger graphs with higher numbers of patrollers.

Table 4.1: Set-partition one-step policy-improvement results on a complete graph, based on 1,000 randomly generated problem scenarios with average inspection times that are comparable to average travel times. Results are reported as the percentage excess over the lower bound for the initial partition set determined from the seed and cluster method, and the expanded partition set obtained from the policy-improvement method.

Number of vertices (n)	Number of patrollers (k)	Percent over z^{LB} (Initial set)	Time (sec)	Percent over z^{LB} (Expanded set)	Time (sec)
10	2	4.83	4.20	2.82	13.67
15	3	6.97	21.34	3.32	70.71
20	4	9.57	48.84	4.16	129.79
25	5	11.40	68.00	4.64	266.86

Very good performance is observed when using the multiple-patroller heuristic method. The results of the baseline experiments are presented in Table 4.2. For the case of 2 patrollers covering 10 vertices, the average result was within 2.82 percent of the lower bound; and for the case of 5 patrollers covering 25 vertices, the average result was within 4.64 percent of the lower bound. Computation time increased significantly as the number of vertices increased. We also note the mean number of pure strategies that were selected by the patrollers for use in the mixed strategy, and the percentage of problems in which the shortest-path strategy was selected in some capacity by the patrollers. The shortest-path strategy was selected in 51.6 percent of the problems for 10 vertices and 2 patrollers,

and in 40.4 percent of the problems with 25 vertices and 5 patrollers.

Table 4.2: Mean performance of the set-partition and shortest-path methods on a complete graph, based on 1,000 randomly generated problem scenarios with average inspection times that are comparable to average travel times, reported as the percentage excess above the lower bound. The mean number of strategies utilized by the patrollers, as well as the percentage of problems that use the shortest-path strategy, are also reported.

Number of vertices (n)	Number of patrollers (k)	Mean number of strategies	Problems using SP (%)	Overall use of SP (%)	Percent over z^{LB} (no SP)	Percent over z^{LB} (with SP)	Time (sec)
10	2	2.79	51.6	9.37	2.99	2.82	10.80
15	3	5.17	48.0	3.68	3.38	3.34	61.44
20	4	6.46	47.2	3.20	4.27	4.16	129.79
25	5	8.84	40.4	1.30	4.67	4.64	266.86

Based on the experimental results, we recommend using both the set-partition with one-step policy-improvement and the shortest-path methods in order to generate a set of pure strategies from which the patrollers can develop a mixed strategy. Exclusive use of the set-partition with one-step policy-improvement method is also very effective, and can be considered for use without significant loss in performance in many problem instances.

4.4.2 Performance on Additional Graph Structures

In addition to problems on a complete graph, we test the multiple-patroller heuristic method on several additional graph structures. Specifically, we consider line graphs, circle graphs, and random trees. We use the procedures from Sections 3.5.4 and 4.4 to generate 1,000 random problem instances for problem cases with n vertices and k patrollers. Results are presented in Table 4.3.

For problems with 15 vertices and 3 patrollers, the multi-patroller heuristic produces on average a result within 9.83 percent of the lower bound for a circle graph and within 15.81 percent of the lower bound for a random tree. The shortest-path method was used the most for a circle graph (82.7 percent of the problems) and the least for a line graph (32.2 percent of the problems). The most different strategies were used for a random tree and the fewest were used for a line graph. These results indicate that the multi-patroller heuristic method can be used effectively on several graph structures as well as sizes.

Table 4.3: Mean performance of the set-partition and shortest-path methods on additional graph structures, based on 1,000 randomly generated problem scenarios for average inspection times that are comparable to average travel times. The mean number of strategies utilized by the patrollers, as well as the percentage of problems that use the shortest-path strategy, are also reported.

Graph structure	Number of vertices (n)	Number of patrollers (k)	Mean number of strategies	Problems using SP (%)	Overall use of SP (%)	Percent over z^{LB}	Time (sec)
Complete	10	2	2.79	51.6	9.37	2.82	10.80
Complete	15	3	5.17	48.0	3.68	3.34	61.44
Line	10	2	1.85	42.5	7.08	10.09	8.16
Line	15	3	2.93	32.2	2.57	14.88	39.06
Circle	10	2	2.94	74.1	17.55	7.10	8.24
Circle	15	3	5.12	82.7	12.60	9.83	30.27
Random tree	10	2	2.83	29.4	2.98	9.90	13.10
Random tree	15	3	5.80	32.8	2.01	15.81	98.88

4.4.3 Sensitivity Analysis

In addition to the baseline case, where average travel times are comparable to average inspection times, we consider the case where the patrollers need to spend more time conducting inspections than they do traveling between vertices, and the case where the patrollers need to spend more time traveling between vertices than they do conducting inspections. The five specific cases we consider in the numerical experiments are the same as the problem cases that were used in Chapters 2 and 3 and are summarized in Table 4.4.

Table 4.4: Numerical experiment cases for multiple patrollers against strategic attackers.

Parameter	Case I	Case II	Case III	Case IV	Case V
Travel time	1×	1×	1×	2×	2×
Inspection time	1×	2×	2×	1×	1×
Attack time	1×	1.5×	1×	1.5×	1×

The mean performance results using the heuristic method in problem cases II through V are presented in Table 4.5. In problem case III, the mean performance was within 0.51 percent of the lower bound in all scenarios. In problem case IV, the mean performance

ranged from within 6.66 percent of the lower bound for the problem of 10 vertices and 2 patrollers, to within 18.11 percent of the lower bound for the problem of 25 vertices and 5 patrollers. These results are similar to the performance observed versus the lower bound in the single patroller versus strategic attackers problem (see Table 3.9).

Table 4.5: Performance of the multi-patroller heuristic mixed strategy based on the combined set-partition and shortest-path strategies. The median number of strategies, $|\Theta|$, considered in both the initial and expanded strategy sets is indicated. Performance is indicated as the percentage excess over the lower bound for the initial and expanded strategy sets.

Case	Number of vertices (n)	Number of patrollers (k)	Initial partition set $\lvert\Theta\rvert$	Percent over z^{LB}	Time (sec)	Expanded partition set $\lvert\Theta\rvert$	Percent over z^{LB}	Time (sec)
II	10	2	2	3.43	4.04	5	1.53	14.32
	15	3	4	4.65	20.09	17	1.53	71.48
	20	4	6	5.76	45.58	25	1.76	139.22
	25	5	9	6.49	62.50	44	1.79	275.26
III	10	2	2	1.34	3.73	5	0.41	13.48
	15	3	4	1.91	18.47	17	0.43	67.43
	20	4	6	2.38	41.54	25	0.50	137.84
	25	5	9	2.65	58.57	44	0.51	274.91
IV	10	2	2	9.05	3.94	5	6.66	13.54
	15	3	4	14.59	20.60	17	9.35	69.66
	20	4	5	24.48	47.87	22	13.93	123.78
	25	5	8	33.82	71.47	38	18.11	239.09
V	10	2	2	2.55	3.88	5	1.56	13.39
	15	3	4	3.57	19.32	17	1.90	71.89
	20	4	6	4.90	43.39	25	2.43	137.92
	25	5	9	5.93	60.29	44	2.76	282.86

CHAPTER 5:
Conclusions and Future Work

5.1 Conclusions

In this dissertation, we examine methods to determine effective patrol policies against both random and strategic attackers. We consider three cases: a single patroller against random attackers, a single patroller against strategic attackers, and multiple patrollers against strategic attackers.

In the case of a single patroller against random attackers, we determine the optimal solution by modeling the state space of the system as a network and solve a minimum cost-to-time ratio cycle problem using linear programming. The solution represents a patrol policy, which is a repeating pattern of locations for a patroller to visit and inspect that minimizes the long-run cost incurred due to undetected attacks. Although the linear program returns the optimal solution, it quickly becomes computationally intractable for problems of moderate size. We therefore develop and test two aggregate-index heuristic methods, the index heuristic time (IHT) method and the index heuristic epoch (IHE) method. Both of these methods consider the structure of the graph, to include travel and inspection time requirements. The IHT method utilizes a predetermined look-ahead time window for the patroller to decide his next action by considering all possible paths and partial paths that can be completed during the time window when starting from his current vertex. For each of these paths, aggregate index values per unit time are computed and the patroller chooses his action based on those index values. He then repeats the process from the next vertex using the same look-ahead time window. This process continues until a patrol pattern is determined. The IHE method works in a similar fashion. However, in this method, a patroller looks ahead a predetermined number of decision epochs, and determines his action by considering all possible paths from the current vertex that consist of the specified number of decision epochs, regardless of the total time those paths will take. We see very favorable results using these methods in numerical experiments. In our baseline experiments, a solution within 1 percent of optimal was returned in at least 90 percent of the problem instances.

In the case of a single patroller against strategic attackers, we determine the optimal

solution by modeling the state space of the system as a network and solve a linear program to minimize the largest expected cost per attack among all vertices. The solution consists of a patrol policy, which is a randomized strategy for the patroller that minimizes the long-run expected cost due to an undetected attack. Although the linear program returns the optimal solution, it quickly becomes computationally intractable for problems of moderate size. We therefore develop two heuristic methods, the shortest-path (SP) and fictitious-play (FP) methods. The SP method uses a combinatorial selection of patrol patterns based on the shortest Hamiltonian cycle in the graph. The FP method is an iterative method based on fictitious play. We also present a linear program that determines a lower bound for the optimal solution, so that we can evaluate our heuristics when the optimal solution is not available. We see very favorable results using both methods in numerical experiments; however, the FP method uses considerably more computation time than the SP method. In our baseline experiments, a solution within 1.2 percent of optimal was returned in at least 90 percent of the problem instances.

Finally, we examine the case of multiple patrollers against strategic attackers, where several patrollers work together to patrol an AOI. In this case, a patrol policy is determined for the entire team and individual patrol policies are then determined for each patroller. The optimal solution can only be determined in a few special cases; therefore, we develop a linear program that determines a lower bound for the optimal solution. We present a heuristic method for the patrollers to develop a mixed strategy by choosing among several pure strategies. We present two methods for the patrollers to determine pure strategies: a method based on vertex set partitions and a method based on the shortest Hamiltonian cycle in the graph. In the set-partition method, the patrol team divides the vertices among the individual patrollers with each patroller then individually executing his best strategy for patrolling the assigned subset of vertices. We present a one-step policy-improvement algorithm that generates effective set partitions based on the heterogeneous properties of each location. In the shortest Hamiltonian cycle method, each patroller uses the same patrol pattern at evenly spaced time intervals. We see favorable results in numerical experiments for several graph structures and patroller combinations when comparing the solution from the heuristic method to the lower bound. For the case of 2 patrollers covering 10 locations, the average result was within 2.82 percent of the lower bound; and for the case of 5 patrollers covering 25 locations, the average result was within 4.64 percent of the lower bound. Computation time increased significantly as the number

of locations increased.

5.2 Future Work

This work provides several areas for continued research. Some of these include a problem formulation which considers the possibility that the patroller may overlook an attacker at the end of an inspection. In other words, the patroller may have less than a 100 percent detection rate of attackers, which may considerably alter the optimal patrol policy. We can also consider the idea of using variable, instead of fixed, inspections times at each location. In this case, a patroller decides how much time to spend inspecting a vertex in order to detect ongoing attacks. For the case of multiple patrollers, the patrollers may consider coordinated efforts beyond the independent vertex set partitions and timed shortest-path patrol strategies.

5.2.1 Inspection with Overlook

We present our problem with the assumption of perfect detection. In other words, there are no false negatives and the patroller will successfully detect all ongoing attacks at a vertex at the end of his inspection. In many practical situations, there may be some probability that a patroller will overlook the presence of an attacker at the end of an inspection. In this case, our model may require significant reformulation to account for this possibility of overlook.

5.2.2 Variable Inspection Times

We can consider situations where the inspection time at a location is not deterministic, but varies according to a probability distribution. This may be applicable if inspection times can vary due to conditions a patroller may encounter during any particular inspection, and this variable effect can be modeled by a distribution. For any continuous distribution of inspection times, we can still formulate our continuous-time model as a semi-Markov decision process, which allows for times between decision epochs to vary according to a probability distribution. If the inspection time at each vertex is exponentially distributed, and thus exhibits the memoryless property, then the system can be modeled more generally as a continuous-time Markov decision process.

A further extension of variable inspection times could involve the patroller choosing how much time he will spend conducting an inspection at a location. A longer inspection time might increase the probability that the patroller will discover an ongoing attack at a

vertex, but a longer inspection time at one vertex will cause the expected cost to increase at the vertices that are not being inspected. This must be considered when determining a patrol policy. In this case, a patroller must not only decide what vertex to visit and inspect next, but must also decide when to end his current inspection in order to move to the next vertex. The optimal choice of follow-on vertex may change based on the amount of time that has passed during the current inspection, which could generate some very complex problem formulations and patrol patterns.

5.2.3 Multiple Coordinated Patrollers

For the multiple-patroller case, our solution consists of a mixed strategy that creates vertex set partitions with each patroller executing his best individual patrol policy or uses a shortest-path patrol pattern with fixed timing. Once the vertices are partitioned and assigned, there is no further coordination among the individual patrollers beyond the continued randomization of the pure strategies. A logical extension of our work is to consider the case where the multiple patrollers coordinate their efforts, perhaps by having multiple patrollers assigned to patrol one or more high-value or high-resource locations to decrease the time between inspections at those locations.

APPENDIX A:
Attack Time Distributions

In this appendix we present the expected value, variance, cumulative distribution function (CDF), and tail distribution functions for common attack time distributions.

A.1 Deterministic Attack Time

Suppose the attack time at a vertex is deterministic with value a. The expected value and variance are

$$\mathrm{E}[X] = a,$$

$$\mathrm{Var}(X) = 0.$$

The CDF is

$$F(t) = P(X < t) = \begin{cases} 0, & t < a; \\ 1, & t \geq a, \end{cases}$$

and

$$\int_0^k F(t)dt = \begin{cases} 0, & k < a; \\ k - a, & k \geq a. \end{cases}$$

The tail distribution function is

$$\bar{F}(t) = P(X > t) = \begin{cases} 1, & t < a; \\ 0, & t \geq a, \end{cases}$$

and

$$\int_0^k \bar{F}(t)dt = \begin{cases} k, & k < a; \\ a, & k \geq a. \end{cases}$$

A.2 Uniform Distribution Attack Time

Suppose the attack time at a vertex follows a uniform distribution with parameters (a, b), with a being the minimum value and b the maximum value. The expected value and variance are

$$\mathrm{E}[X] = \frac{a+b}{2},$$

$$\mathrm{Var}(X) = \frac{(b-a)^2}{12}.$$

The CDF is

$$F(t) = P(X < t) = \begin{cases} 0, & t < a; \\ \frac{t-a}{b-a}, & a \le t < b; \\ 1, & t \ge b, \end{cases}$$

and

$$\int_0^k F(t)dt = \begin{cases} 0, & k < a; \\ \frac{(k-a)^2}{2(b-a)}, & a \le k < b; \\ k - \frac{a+b}{2}, & k \ge b. \end{cases}$$

The tail distribution function is

$$\bar{F}(t) = P(X > t) = \begin{cases} 1, & t < a; \\ \frac{b-t}{b-a}, & a \le t < b; \\ 0, & t \ge b, \end{cases}$$

and

$$\int_0^k \bar{F}(t)dt = \begin{cases} k, & k < a; \\ a + \frac{(k-a)(2b-k-a)}{2(b-a)}, & a \le k < b; \\ \frac{a+b}{2}, & k \ge b. \end{cases}$$

A.3 Triangular Distribution Attack Time

Suppose the attack time at a vertex follows a triangular distribution with parameters (a, b, c), with a being the minimum value, b the maximum value, and c the mode. The expected value and variance are

$$\mathrm{E}[X] = \frac{a+b+c}{3},$$

$$\mathrm{Var}(X) = \frac{a^2+b^2+c^2-ab-ac-bc}{18}.$$

The CDF is

$$F(t) = P(X < t) = \begin{cases} 0, & t < a; \\ \frac{(t-a)^2}{(b-a)(c-a)}, & a \le t < c; \\ 1 - \frac{(b-t)^2}{(b-a)(b-c)}, & c \le t < b; \\ 1, & t \ge b, \end{cases}$$

and

$$\int_0^k F(t)dt = \begin{cases} 0, & k < a; \\ \frac{(k-a)^3}{3(b-a)(c-a)}, & a \le k < c; \\ k - \frac{a+b+c}{3} + \frac{(b-k)^3}{3(b-a)(b-c)}, & c \le k < b; \\ k - \frac{a+b+c}{3}, & k \ge b. \end{cases}$$

The tail distribution function is

$$\bar{F}(t) = P(X > t) = \begin{cases} 1, & t < a; \\ 1 - \frac{(t-a)^2}{(b-a)(c-a)}, & a \le t < c; \\ \frac{(b-t)^2}{(b-a)(b-c)}, & c \le t < b; \\ 0, & t \ge b, \end{cases}$$

and

$$\int_0^k \bar{F}(t)dt = \begin{cases} k, & k < a; \\ k - \frac{(k-a)^3}{3(b-a)(c-a)}, & a \le k < c; \\ \frac{a+b+c}{3} - \frac{(b-k)^3}{3(b-a)(b-c)}, & c \le k < b; \\ \frac{a+b+c}{3}, & k \ge b. \end{cases}$$

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B:
Generation of Problem Instances

To generate a random graph of locations for our experiments, let (X_i, Y_i) denote the Cartesian coordinate of vertex i, for $i \in N$, and draw X_i and Y_i from independent uniform distributions over $[0, 1]$. Letting d_{ij} denote the travel distance between vertices i and j, we compute

$$d_{i,j} = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}, \quad \forall i, j \in N.$$

We determine the expected value for $d_{i,j}$ by computing the expected values of the quantities $(X_i - X_j)^2$ and $(Y_i - Y_j)^2$. Since the random variables are independent, these two quantities will have the same expected value and we only need to compute $\mathrm{E}[(X_i - X_j)^2]$ as follows,

$$\begin{aligned}\mathrm{E}[(X_i - X_j)^2] &= \mathrm{E}[X_i^2 - 2X_iX_j + X_j^2] \\ &= \mathrm{E}[X_i^2] - 2\mathrm{E}[X_iX_j] + \mathrm{E}[X_j^2],\end{aligned}$$

which due to independence can be expressed as

$$= \mathrm{E}[X_i^2] - 2\mathrm{E}[X_i]\mathrm{E}[X_j] + \mathrm{E}[X_j^2].$$

In this case, $\mathrm{E}[X_i] = \frac{1}{2} = \mathrm{E}[X_j]$ and $\mathrm{E}[X_i^2] = \frac{1}{3} = \mathrm{E}[X_j^2]$. Therefore,

$$\mathrm{E}[(X_i - X_j)^2] = \frac{1}{3} - 2\Big(\frac{1}{2}\Big)\Big(\frac{1}{2}\Big) + \frac{1}{3} = \frac{1}{6} = \mathrm{E}[(Y_i - Y_j)^2],$$

and

$$\mathrm{E}[d_{ij}^2] = \frac{1}{6} + \frac{1}{6} = \frac{1}{3}.$$

Although we cannot determine $\mathrm{E}[d_{ij}]$ in closed form, we know that when $\mathrm{Var}(d_{ij}) \neq 0$,

$$\mathrm{E}[d_{ij}] < \sqrt{\mathrm{E}[d_{ij}^2]} = \frac{1}{\sqrt{3}} \approx 0.57735.$$

We conduct a simulation to determine a value for $\mathrm{E}[d_{ij}]$ by generating 1,000,000 independent sets of four random variables distributed over $\mathrm{U}[0, 1]$. Using these values, we determine the mean and variance of d_{ij} to be $\mathrm{E}[d_{ij}] = 0.5215$ and $\mathrm{Var}(d_{ij}) = 0.0615$.

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

Ahuja, R., T. Magnanti, J. Orlin. 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ.

Almeida, A., G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, Y. Chevaleyre. 2004. Recent advances on multi-agent patrolling. *Advances in Artificial Intelligence–SBIA 2004*. Springer-Verlag, Berlin, Germany, 474–483.

Alpern, S. 1992. Infiltration games on arbitrary graphs. *Journal of Mathematical Analysis and Applications* **163**(1) 286–288.

Alpern, S. 2010. Search games on trees with asymmetric travel times. *SIAM Journal on Control and Optimization* **48**(8) 5547–5563.

Alpern, S., R. Fokkink. 2008. *Accumulation games on graphs - LSE-CDAM-2008-18*. London School of Economics, London, U.K.

Alpern, S., S. Gal. 2002. Searching for an agent who may or may not want to be found. *Operations Research* **50**(2) 311–323.

Alpern, S., A. Morton, K. Papadaki. 2011. Patrolling games. *Operations Research* **59**(5) 1246–1257.

Auger, J. 1991. An infiltration game on k arcs. *Naval Research Logistics* **38**(4) 511–529.

Avenhaus, R. 2004. Applications of inspection games. *Mathematical Modeling and Analysis* **9**(3) 179–192.

Basilico, N., N. Gatti, F. Amigoni. 2009. Developing a deterministic patrolling strategy for security agents. *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2. IEEE Computer Society, Milan, Italy, 565–572.

Benkoski, S., M. Monticino, J. Weisinger. 1991. A survey of the search theory literature. *Naval Research Logistics* **38** 469–464.

Bóna, M. 2011. *A Walk through Combinatorics: An Introduction to Enumeration and Graph Theory,* 3rd ed. World Scientific, Hackensack, NJ.

Chevaleyre, Y. 2004. Theoretical analysis of the multi-agent patrolling problem. *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*. IEEE Computer Society, Beijing, China, 302–308.

Elor, Y., A. Bruckstein. 2009. Multi-a (ge) nt graph patrolling and partitioning. *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, vol. 2. IEEE Computer Society, Milan, Italy, 52–57.

Garnaev, A., G. Garnaeva, P. Goutal. 1997. On the infiltration game. *International Journal of Game Theory* **26**(2) 215–221.

Gittins, J., K. Glazebrook, R. Weber. 2011. *Multi-armed Bandit Allocation Indices,* 2nd ed. Wiley, Hoboken, NJ.

Kikuta, K. 1995. A search game with traveling cost on a tree. *Journal of the Operations Research Society of Japan* **38**(1) 70–88.

Kikuta, K., W. Ruckle. 1994. Initial point search on weighted trees. *Naval Research Logistics* **41** 821–831.

Kikuta, K., W. Ruckle. 2002. Continuous accumulation games on discrete locations. *Naval Research Logistics* **49**(1) 60–77.

Lin, K., M. Atkinson, T. Chung, K. Glazebrook. 2013. A graph patrol problem with random attack times. *Operations Research* **61**(3) 694–710.

Machado, A., G. Ramalho, J. Zucker, A. Drogoul. 2003. Multi-agent patrolling: An empirical analysis of alternative architectures. *Multi-Agent-Based Simulation II*. Springer-Verlag, Berlin, Germany, 155–170.

Owen, G. 1995. *Game Theory,* 3rd ed. Academic Press, San Diego, CA.

Paruchuri, P., J. Pearce, M. Tambe, F. Ordóñez, S. Kraus. 2007. An efficient heuristic approach for security against multiple adversaries. *Proceedings of the 6th international joint conference on autonomous agents and multiagent systems*. International Foundation for Autonomous Agents and Multiagent Systems, Honolulu, HI, 181–188.

Paruchuri, P., M. Tambe, F. Ordóñez, S. Kraus. 2006. Security in multiagent systems by policy randomization. *Proceedings of the 5th international joint conference on autonomous agents and multiagent systems*. International Foundation for Autonomous Agents and Multiagent Systems, Hakodate, Japan, 273–280.

Portugal, D., R. Rocha. 2011. A survey on multi-robot patrolling algorithms. *Technological Innovation for Sustainability*. Springer-Verlag, Berlin, Germany, 139–146.

Puterman, M. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience, New York, NY.

Robinson, J. 1951. An iterative method of solving a game. *Annals of Mathematics* **54**(2) 296–301.

Ross, S. 2010. *Introduction to Probability Models,* 10th ed. Academic Press, San Diego, CA.

Ruckle, W. 1983. *Geometric Games and their Applications*. Pitman, Boston, MA.

Sak, T., J. Wainer, S. Goldenstein. 2008. Probabilistic multiagent patrolling. *Advances in Artificial Intelligence-SBIA 2008*. Springer-Verlag, Berlin, Germany, 124–133.

Tambe, M. 2012. *Security and Game Theory*. Cambridge University Press, New York, NY.

Washburn, A. 2003. *Two-Person Zero-Sum Games,* 3rd ed. INFORMS, Linthicum, MD.

Washburn, A., K. Wood. 1995. Two-person zero-sum games for network interdiction. *Operations Research* **43**(2) 243–351.

Zoroa, K., P. Zoroa, M. Fernandez-Saez. 2009. Weighted search games. *European Journal of Operations Research* **195**(2) 394–411.

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California